

[Docs](#) » Dev documentation

## Welcome to the ArduPilot Development Site

Tip

Keep up with the latest ArduPilot related blogs on [ArduPilot.org](#)!

ArduPilot/APM is a open source autopilot system supporting multi-copters, traditional helicopters, fixed wing aircraft and rovers. The source code is developed by a [large community of enthusiasts](#). New developers are always welcome! The best way to start is by joining the [Developer Team Forum](#), which is open to all and chock-full of daily development goodness. Lurk for a while to get a feel for it, then participate!

### Why the name?

The ‘Ardu’ part of the ArduPilot name comes from Arduino. The original APM1 autopilot board was based around the [Arduino](#) development environment. We’ve since outgrown the Arduino environment and no longer use the Arduino runtime libraries, although we do still support building the ArduPilot for the AVR based APM1 and APM2 boards using a slightly modified version of the Arduino integrated development environment. A timeline history of ArduPilot can be found [here](#).

### Supported boards

[Supported AutoPilot Controller Boards](#) provides an overview and key links for all the supported controller boards, including [Pixhawk](#), [Arsov UAV-X2](#), [Erle-Brain](#), [NAVIO+](#) etc.

The ArduPilot/APM source code is written on top of the [AP-HAL](#) hardware abstraction layer, making it possible to port the code to a wide range of autopilot boards. See this [blog post](#) for more information on the move to AP-HAL.

### Project List

The ArduPilot system is made up of (or relies upon) several different projects which are listed below. Those marked with an asterix (\*) are peer projects that have their own owners outside the core ArduPilot dev team.

- DroneKit ([site](#)) - APM SDK for apps running on vehicles, mobile devices and/or in the cloud.
- Plane ([wiki](#), [code](#)) - autopilot for planes
- Copter ([wiki](#), [code](#)) - autopilot for multicopters and traditional helicopters
- Rover ([wiki](#), [code](#)) - autopilot for ground vehicles
- Mission Planner ([wiki](#), [code](#)) - the most commonly used ground station written in C# for windows but also runs on Linux and MacOS via mono
- APM Planner 2.0 ([wiki](#), [code](#)) is a ground station specifically for APM written in C++ using the Qt libraries
- MAVProxy ([wiki](#)) - command line oriented and scriptable ground station (mostly used by developers)
- MinimOSD ([wiki](#), [code](#)) - on-screen display of flight data

- AndroPilot ([user guide](#), [code](#), [google play](#)) - android ground station
- DroneAPI ([tutorial](#), [droneshare](#)) - A developer API for drone coprocessors and web applications.
- DroidPlanner2 ([wiki](#), [code](#), [google play](#)) - android ground station
- [QGroundControl](#) is an alternative ground station written in C++ using the Qt libraries
- PX4 ([wiki](#)) - designers of the PX4FMU and owners of the underlying libraries upon which Plane/Copter/Rover use when running on the PX4FMU
- MAVLink ([wiki](#)) - the protocol for communication between the ground station, flight controller and some peripherals including the OSD. A “Dummy’s Guide” to working with MAVLink is [here](#).

## Getting Started with ArduPilot Development

The main entry points for developing flight controller/antenna tracker and companion computer code are listed in the sidebar.

For topics related to Ground Control Station development see:

- [Building Mission Planner \(C#, Windows\)](#)
- [Building APM Planner 2.0 \(Qt, C++, Linux, Mac OSX, Windows\)](#)

## RTF vehicles

- [3DR Solo](#)

## How the team works

- [Bringing new members onto the team](#)
- The main developer discussion mailing list is [drones-discuss](#), and is open to anyone to join
- The development team is also using [gitter](#) for APM development discussions - <https://gitter.im/ArduPilot/ardupilot>
- We have a [mumble server](#) for real-time voice discussions
- Our annual developers conference is [DroneCon](#). See previous years speeches and content [here](#).
- The source code for ArduPilot/APM is managed using git on <https://github.com/ArduPilot/ardupilot>
- Pre-compiled firmware for supported autopilot boards is available from <http://firmware.ardupilot.org>
- User support is available on the [APM forums](#).
- The ArduPilot/APM [automatic test system](#) shows the test status of each commit. It's described [here](#).
- Bug tracking and open issues are tracked using the [github issues system](#)
- Vehicle onboard parameter documentation for [copter](#), [plane](#) and [rover](#) is auto-generated from the source code
- [Release Procedures for Copter](#)
- [Current and Past Dev Team members](#)

## Development languages and tools

The main flight code for ArduPilot is written in C++. Support tools are written in a variety of languages, most commonly in python. Currently the main vehicle code is written as '.pde' files, which come from the Arduino build system. The pde files are preprocessed into a .cpp file as part of the build. The include statements in the pde files also provide implied build rules for what libraries to include and link to.

## License

ArduPilot (including Copter, Plane, Rover and MissionPlanner) is released as free software under the [GNU General Public License](#) version 3 or later. See [License overview wiki page here](#).

## Didn't find what you are looking for?

If you think of something that should be added to this site, please [open an issue](#) or post a comment on the [drones-discuss](#) mailing list.

# Full Table of Contents

## Working with the ArduPilot Project Code

This article explains where to get the ArduPilot code and how to submit changes to the project.

### Overview

The ArduPilot project uses [git](#) for source code management and [GitHub](#) for source code hosting.

Developers who want to contribute to ArduPilot will fork the project, create a branch on their fork with new features, and then raise a pull request to get the changes merged into the “master” project.

Developers who just want to use and build the latest code can do so by cloning and building the “master” repository.

The ArduPilot project source code for Plane, Copter, Rover and Antenna Tracker are available on [GitHub](#) in the <https://github.com/ArduPilot/ardupilot> repository. Several additional projects are used for PX4 based platforms (ie. PX4v1 and Pixhawk): [PX4Firmware](#), [PX4NuttX](#), [uavcan](#) — these are imported as Git Submodules when you build the project.

MissionPlanner is in the [ardupilot/MissionPlanner](#) repository.

#### Note

An older Google Code repository remains online for legacy reasons, but unless you specifically need older (APM 1.x) resources, you won’t need to use it.

### Prerequisites

The ArduPilot project uses [git](#) for source code management.

Git is available on all major OS platforms, and a variety of tools exist to make it easier to get started. First, you need to [download and install a client for your operating system](#). If you’re new to source code control systems, the [GitHub for Windows](#) or [GitHub for Mac](#) clients are well-documented and integrate well with GitHub and are a good place to start. This guide will use both the GitHub for Windows user interface as well as the command-line interface through an OSX/Linux Terminal.

If you are working towards submitting code back to the official APM source code repository, you’ll need to [sign up for a free user account with Github](#).

### Learning git

This guide covers the basic git commands/concepts needed to work with the project: clone, branch, commit, push.

If you want to know more about git there are many great resources online. Here are just a few you may find useful:

- [Try Git](#): browser-based interactive tutorial for learning git
- [Git Ready](#): tutorials of varying difficulty levels
- [Git SCM Book](#): introduction and full documentation

### Forking the main repository

#### Tip

If you just want to build and test the project source code (without making changes) you can skip this step and just clone the main project repository (next section).

“Forking” is GitHub’s term for copying a repository to your own account. The forked repository preserves information about the original project so you can fetch updates from it (and contribute changes back to it). If you want to contribute changes back to the main project you will need to first create your own fork of the main ArduPilot repository.

To fork the main repository:

- Log into Github and go to <https://github.com/ArduPilot/ardupilot>.
- At the upper right is a button, “Fork”:



Click the **Fork** button and follow the directions.

When your are finished there will be a new repository within your account:

`//github.com/your-github-account-name/ardupilot`

This forked repository is what you will clone and work on locally when making changes to the code.

## Cloning the repository

“Cloning” is git’s term for making a copy of any repository on your own computer. You can clone either your own fork of the repository (if you want to make changes to the source code) or the main ArduPilot repository.

The information/tools you need for cloning the project are on the right side of the screen on each GitHub repository home page.

## HTTPS clone URL

<https://github.com>



You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).



[Clone in Desktop](#)



[Download ZIP](#)

### **Github section for cloning a repo**

#### **OSX/Linux Terminal:**

- Open a Terminal and navigate to the directory where you would like to clone the project
- Clone your fork:

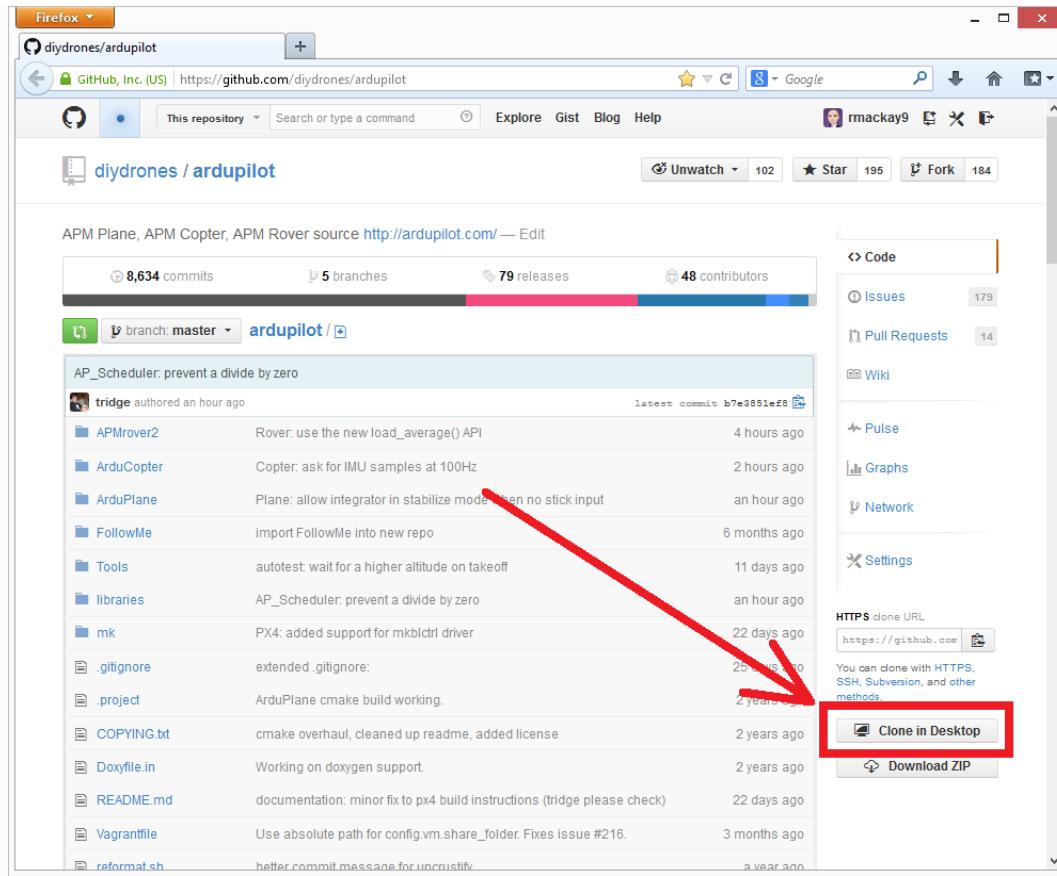
```
git clone https://github.com/your-github-account-name/ardupilot
cd ardupilot
git submodule update --init --recursive
```

or the main project:

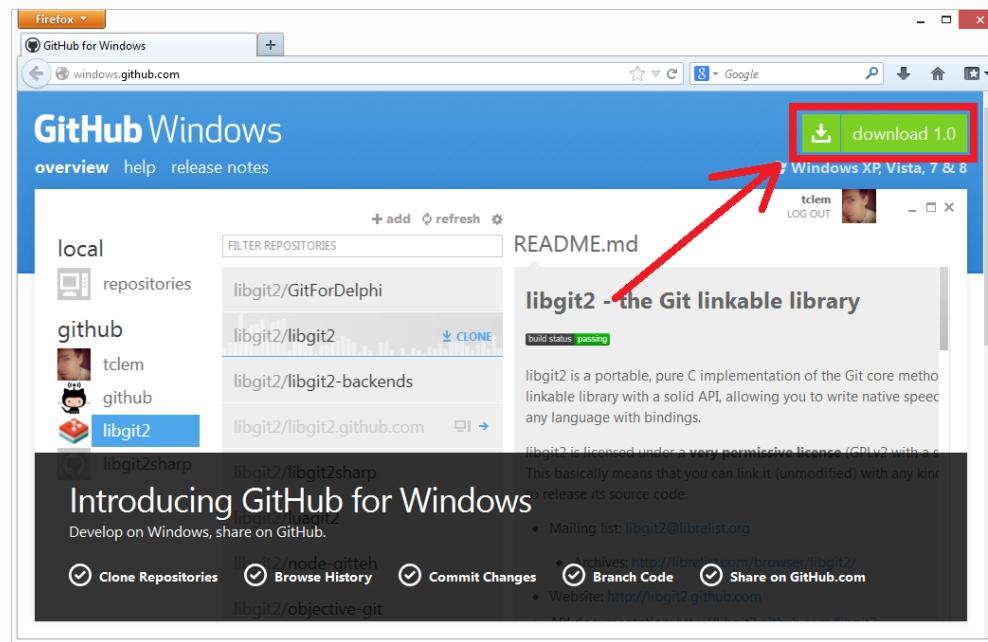
```
git clone https://github.com/ArduPilot/ardupilot
cd ardupilot
git submodule update --init --recursive
```

#### **Windows (GitHub GUI):**

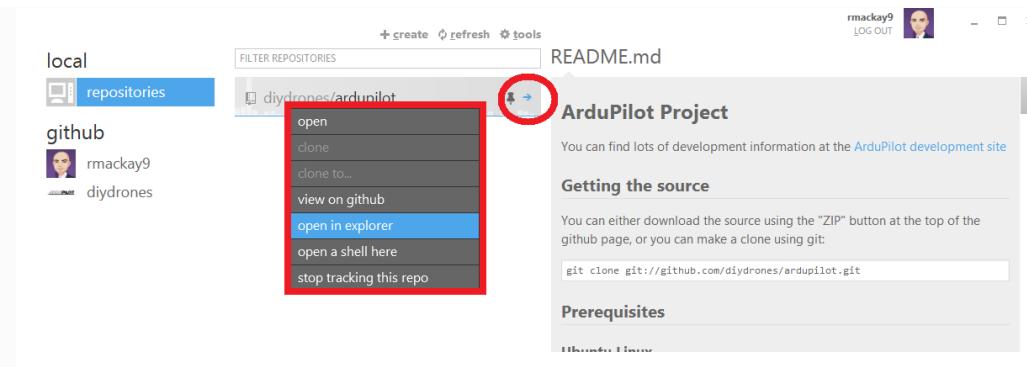
- Open the [ardupilot/ardupilot](#) repository in your favorite web browser
- Click on the “Clone in Desktop” button on the bottom right



- If you have not installed GitHub before:
  - When brought to the windows.github.com page, push the “download” button on the top right



- Save the **GitHubSetup.exe** somewhere on your machine and then run it and follow the instructions to install GitHub client
- On GitHub client click the right arrow button to view a list of recent commits or right-mouse-button click on the ardulibot/ardulibot repository and “open in explorer”.



- You can now also open the file in your favourite editor such as [NotePad++](#), [Sublime Text](#) or [acme](#).

## Building the code

ArduPilot supports building many different build targets (vehicles and autopilot hardware) on Linux, Windows and Mac OSX. For information about how to build for your particular target, see [Building the code](#).

## Making a branch and changing some code

Branches are a way to isolate different paths of development, which can then be combined in a single (often named “master”) branch. Refer to [this short guide](#) for more information, in particular the resources under the [Learning Git](#) section. In this section of the tutorial, you’ll make a branch and change some code.

Branch names are up to you, but it can be helpful to choose short descriptive names. The branch name used for this tutorial is “apm\_git\_tutorial”

### OSX/Linux Terminal commands

These commands assume your current working directory is the root of the repository you cloned.

#### Tip

These same commands can be used in Windows too if you use a command line git client (e.g. The “Git Shell” utility that was installed with GitHub for Windows).

1. Create a branch.

```
git checkout -b apm_git_tutorial
```

2. Change some code. For this tutorial, open the **Tools/GIT\_Test/GIT\_Success.txt** in your preferred text editor, and put your name at the end of the file. Save the file.

3. See that you’ve changed some files by checking the status:

```
git status
```

4. Commit your work to the branch to add your changes to the git history:

```
git add Tools/GIT_Test/Git_Success.txt
git commit -m 'Added name to GIT_Success.txt'
```

Please see below for further information regarding conventions for committing work that you expect to be integrated into the official releases. When you commit, you’re required to add a log message explaining what you did in the commit. See [Submitting Patches Back to Master](#) for more information on how to do this, and for the purpose of this tutorial, you can just use a single line stating: “Added

name to GIT\_Success.txt":

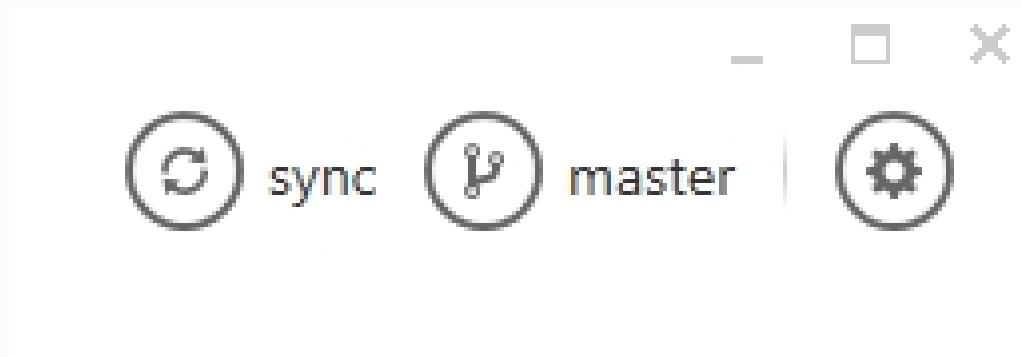
- Push your branch to GitHub. This will copy your work on your local branch to a new branch on GitHub. Pushing branches is a precondition for collaborating with others on GitHub or for submitting patches back to the official releases. It is assumed origin is the remote name of your fork of the github repository.

```
git push origin HEAD:apm_git_tutorial
```

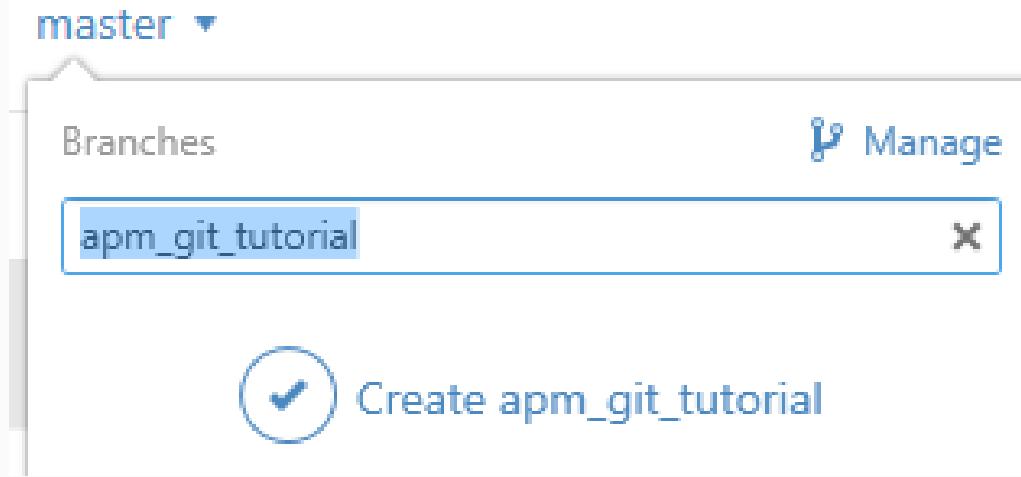
### Windows (GitHub GUI)

In the GitHub GUI you used to clone the repository, you can create a branch and commit it.

- Create a branch. In the GitHub for Windows application, click on the 'master' button in the upper right corner of the window.

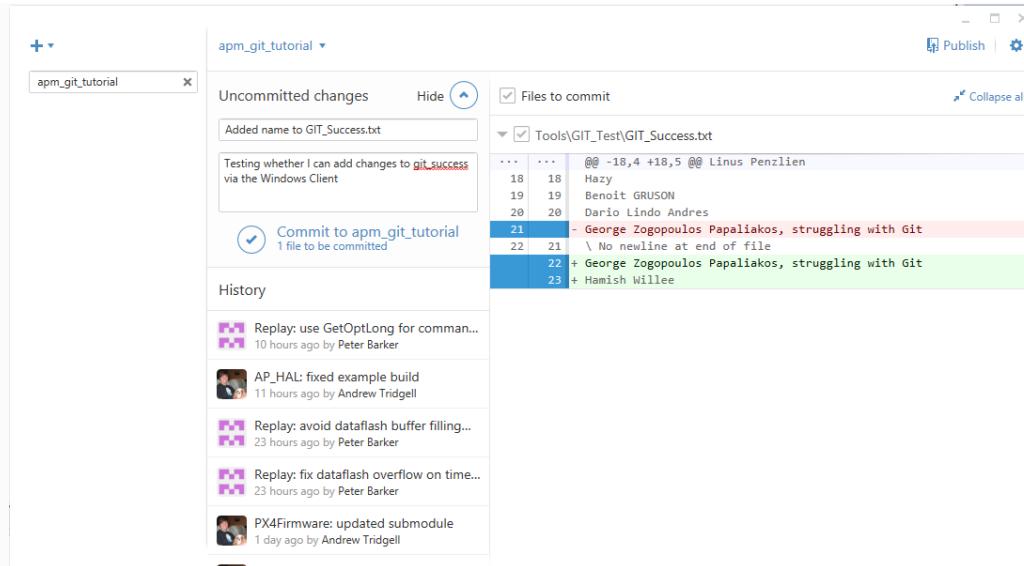


Enter 'apm\_git\_tutorial' and click the "+ create branch: apm\_git\_tutorial" dropdown.



### **Create a new branch in Github for Windows**

- Change some code. For this tutorial, open the **Tools/GIT\_Test/GIT\_Success.txt** in your preferred text editor, and put your name at the end of the file. Save the file.
- The Git for Windows client shows the changed file and has a place where you can enter a summary and description of the change. For the purpose of this tutorial, you can just use a single line stating: "Added name to GIT\_Success.txt"



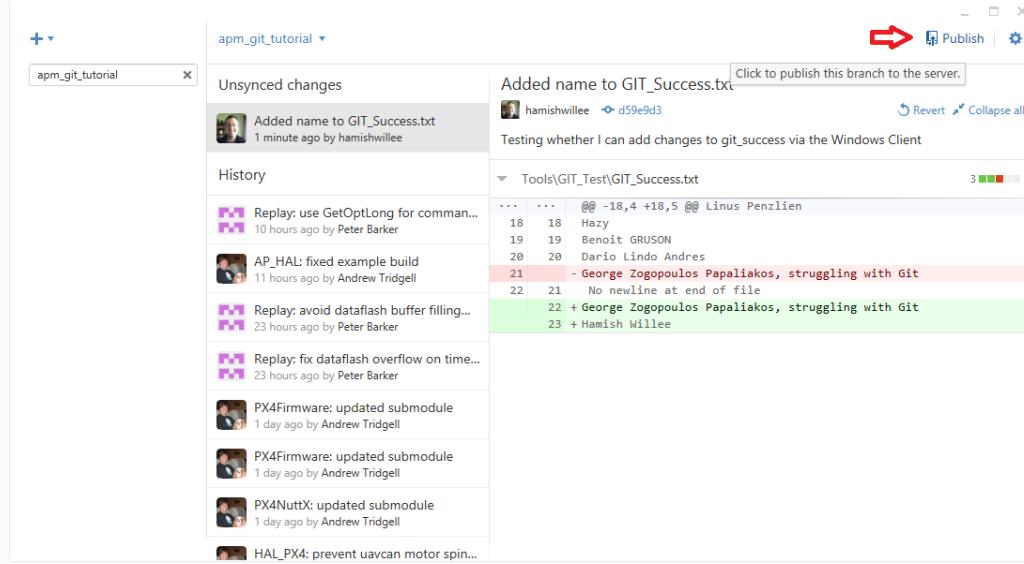
## ***Github for windows: Commit Change***

### Note

Please see [Submitting Patches Back to Master](#) for further

information regarding conventions for committing work that you expect to be integrated into the official releases (this will include a much more detailed commit message).

4. Commit your work to the branch by pressing the **Commit to apm\_git\_tutorial** link.
5. Push your local branch to GitHub (pushing branches is a precondition for collaborating with others on GitHub or for submitting patches back to the official releases). In the client you can do this by pressing the **Publish** link:



## ***GitHub for Windows Client: Pushing changes***

Congratulations! This is bulk of the normal process you'd follow when working on code to submit back to the official project. The next step is to [submit a pull request](#) so your changes can be considered for addition to the main project.

### **Rebase-based workflow: keeping your code up to date**

As you develop, the (original) master branch of the ArduPilot repository is likely to be updated, and you should [keep your fork and your local branches up to date](#). Rebasing allows you to re-apply your changes on top of the latest version of the original repo, making it much easier for the project to merge them.

The following commands can be used to rebase your fork of the project to the “upstream master” (main project repo). You can enter these commands direct into a Linux/OSX Terminal. If using GitHub for Windows, launch the “Git Shell” utility that was installed with GitHub for Windows application.

1. Navigate to your ardupilot git repository.

```
cd <ardupilot-path>
```

2. Ensure you are looking at your master branch

```
git checkout master
```

3. Ensure your repository is connected to the upstream repository you forked from.

```
git remote add upstream https://github.com/ArduPilot/ardupilot.git
```

4. Fetch changes from the upstream master.

```
git fetch upstream
```

5. Rebase your current branch from the upstream master.

```
git rebase upstream/master
```

6. Ensure your repository is connected to the your repository on github.

```
git remote add origin https://github.com/your-github-account-name/ardupilot.git
```

7. Now push the updated master to your github repository

```
git push origin master
```

## Working with git submodules

ArduPilot development for PX4 based platforms (ie. PX4v1 and Pixhawk) uses three additional repositories:

- [PX4Firmware](#)
- [PX4NuttX](#)
- [uavcan](#)

These are *Git submodules* of the ArduPilot project, and are automatically fetched as part of a build when needed. For more information on working with these projects see [Git Submodules](#).

## License (GPLv3)

ArduPilot (including Copter, Plane, Rover and AntennaTracker) and the Mission Planner are free software: you can redistribute it and/or modify it under the terms of the GNU General Public License version 3 as published by the [Free Software Foundation](#).

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For more specific details see <http://www.gnu.org/licenses>, the [Quick Guide to GPLv3](#). and the [copying.txt](#) in the codebase.

The GNU operating system which is under the same license has an informative [FAQ here](#).

## Note to developers

We very much appreciate you using our code to do fun and interesting things with. We hope that while doing this you may find and fix bugs or make enhancements that could be useful for the greater community and will make the developers aware of them by emailing [drones-discuss@googlegroups.com](mailto:drones-discuss@googlegroups.com) and/or [contribute them back using a pull request](#) so they can be considered to be added to the original code base.

## Note to businesses or individuals including this software in products

We also greatly appreciate those companies and individuals who incorporate this software into their products for sale. A significant number have already done this. There are some things that the license requires however that we need to point out:

- Inform your customers that the flight code software is open source and provide the actual source code in the product or provide a link to where the source code can be found (see sample below).

PROJECT	COMPONENT	VERSION	LICENSE
ArduPilot Copter	Flight code	3.4	GPLv3
MAVproxy		1.4.4	GPLv3
pyMAVlink		1.4.41	LGPLv3
DroneKit		1.4	Apache
Pixhawk	Autopilot hardware	2	CC BY-SA 3.0

Source: <https://github.com/ArduPilot/ardupilot>

- As with the contributions of individual developers, we would like it if you could keep us informed of the products that incorporate the software by emailing [drones-discuss@googlegroups.com](mailto:drones-discuss@googlegroups.com). Also for those changes that could be useful to the wider community we would appreciate it if you could [contribute them back using a pull request](#) for consideration to be added to the original code base.

To both individual developers and companies we also ask that when making derivative works, the original credits listing all the individuals that contributed to getting the software to its current form are left in place.

## Why did we pick this license over others?

- the requirement to contribute back bug fixes and enhancements to the project (or at least provide those fixes to the end customer) increases cooperation amongst the contributors. Without this requirement participants are tempted to keep even small improvements to themselves in order to gain an advantage over other contributors. There is evidence that this quickly leads to many incompatible forks of the project to the detriment of all.

Linus Torvalds (inventor of Linux) agrees.

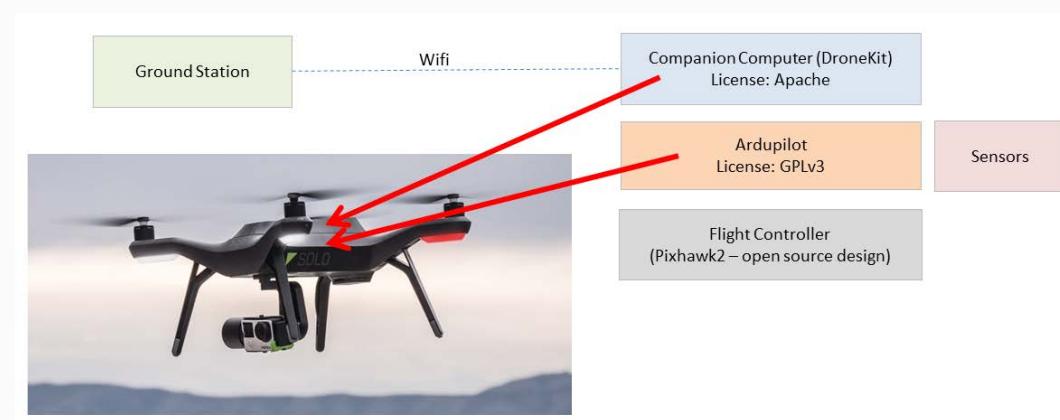
*"I love GPL v2. I really think the license has been one of the defining factors in the success of Linux because it enforced that you have to give back, which meant that the fragmentation has never been something that has been viable from a technical standpoint."*

-Linux Torvalis, LinuxCon 2016

- the “v3” portion of the license ensures that the customer who purchased the vehicle has the right to upgrade or replace the version of ArduPilot on the flight controller. The license doesn’t require that it actually work but just that the upgrade is possible. This ensures that even if a manufacturer stops supporting the product (which can happen for very valid reasons) the product can continue to be useful if the owner or a community of developers decides to pick up support. Examples of this have already happened with ArduPilot.

## Can I integrate Closed source (i.e. Proprietary) and Open Source?

Ardupilot is open source (GPLv3) but you can use a companion computer to run closed source code to ease integrating ArduPilot into your corporate systems or to add higher level features to differentiate yourself from your rivals. You build on the reliability of the free low-level flight code so you can instead invest in the higher level features. We believe ArduPilot is as reliable as the leading closed system and you are not beholden to a particular manufacturer. Below is an image of how one manufacturer accomplished this.



## Supported AutoPilot Controller Boards

Currently ArduPilot supports the following autopilot boards.

### Pixhawk

Next-generation PX4, with more memory, improved sensors and a much easier-to-use design. See the [product overview](#) (store) and [Pixhawk Overview](#) (wiki) for more information.

<b>Purchase</b>	<a href="http://store.3drobotics.com">store.3drobotics.com</a>
	<b>Product Description</b>
	<b>OS:</b>
	NuttX
	<b>CPU:</b>
	32-bit STM32F427 Cortex M4 core with FPU
	32 bit STM32F103 failsafe co-processor
	<b>Memory:</b>

	<p>168 MHz/256 KB RAM/2 MB Flash</p> <p><b>Sensors:</b></p> <p>ST Micro L3GD20 3-axis 16-bit gyroscope</p> <p>ST Micro LSM303D 3-axis 14-bit accelerometer / compass (magnetometer)</p> <p>Invensense MPU 6000 3-axis accelerometer/gyroscope</p> <p>MEAS MS5611 barometer</p>
<b>Specifications</b>	<p>5x UART (serial ports), one high-power capable, 2x with HW flow control</p> <p>2x CAN</p> <p>Spektrum DSM / DSM2 / DSM-X® Satellite compatible input up to DX8 (DX9 and above not supported)</p> <p>Futaba S.BUS® compatible input and output</p> <p>PPM sum signal</p> <p>RSSI (PWM or voltage) input</p> <p>I2C®</p> <p>SPI</p> <p>3.3 and 6.6V ADC inputs</p> <p>External microUSB port</p>
	<p><b>Power System:</b></p> <p>Ideal diode controller with automatic failover</p> <p>Servo rail high-power (7 V) and high-current ready</p> <p>All peripheral outputs over-current protected, all inputs ESD protected</p>
	<p><b>Weight and Dimensions:</b></p> <p>Weight: 38g (1.31oz)</p> <p>Width: 50mm (1.96")</p> <p>Thickness: 15.5mm (.613")</p> <p>Length: 81.5mm (3.21")</p>
<b>Setup</b>	<a href="#">Pixhawk Overview</a> , <a href="#">Powering</a>
<b>Design files</b>	<a href="#">Schematic</a> <a href="#">Layout</a>

## APM2.x

[APM2](#) is a popular AVR2560 8-bit autopilot.

#### Note

APM 2.6 is compatible with Copter 3.2 and earlier only (to use APM:Copter version 3.3 and later, please purchase a Pixhawk). Plane and Rover have full support for APM 2.6 in all existing releases. This board is not recommended for any new users.

<b>Purchase</b>	<a href="#">store.drones.com</a>
	<b>OS:</b> None
<b>Specifications</b>	<b>CPU:</b> AtMega 2560  <b>Memory:Sensors:</b> 3-axis gyro, accelerometer High-performance Barometric pressure sensor MS5611-01BA03  <b>Interfaces:</b>  <b>Power System:</b> <a href="#">Archived:APM 2.5 and 2.6 Overview</a>
<b>Setup</b>	<a href="#">Archived:APM 2.5 and 2.6 Overview</a>
<b>Design files</b>	<a href="#">APM schematic diagram</a> <a href="#">APM board layout</a>

## PX4

A 32 bit ARM based autopilot with many advanced features, using the [NuttX](#) real-time operating system. See the [PX4 Overview](#) (wiki) for more information.

<b>Purchase</b>	Not available
	<b>OS:</b> NuttX  <b>CPU:</b>

	ARM Cortex-M4F microcontroller running at 168MHz with DSP and floating-point hardware acceleration.
Specifications	<p><b>Memory:</b></p> <p>1024KB of flash memory, 192KB of RAM</p> <p><b>Sensors:</b></p> <p>MEMS accelerometer and gyro, compass and barometric pressure sensor.</p>
	<p><b>Interfaces:</b></p> <p>?</p>
	<p><b>Power System:</b></p> <p>?</p>
	<p><b>Weight and Dimensions:</b></p> <p>Weight: 8.10 g</p> <p>Width:</p> <p>Thickness:</p> <p>Length:</p>
Setup	<a href="#">Archived:PX4FMU Overview</a>
Design files	<p><a href="#">Module homepage</a></p> <p><a href="#">Manual</a></p> <p><a href="#">Schematics download</a></p> <p><a href="#">Eagle files for version 1.6 download</a></p> <p><a href="#">Eagle files for version 1.7 download</a></p>

## Arsov UAV-X2

Arsov UAV-X2 is a high quality, compact, light weight and cost effective alternative to the PX4 V2 or PixHawk autopilots. It is 100% compatible with the PX4 firmware.

Purchase	<a href="http://www.auav.co">www.auav.co</a>
	<p><b>OS:</b></p> <p>NuttX</p> <p><b>CPU:</b></p> <p>STM32F427VI ARM microcontroller</p> <p>STM32F100C8T6 ARM microcontroller</p> <p><b>Memory:</b></p> <p><b>Sensors:</b></p>

<b>Specifications</b>	<p>Gyroscope: ST Micro L3GD20</p> <p>Accelerometer: ST Micro LSM303D</p> <p>Gyro: Invensense MPU 6000</p> <p>MEAS MS5611 barometer</p> <p><b>Interfaces:</b></p> <ul style="list-style-type: none"> <li>3 x UART</li> <li>1 x CAN</li> <li>1 x I2C</li> <li>1 x SPI</li> <li>2 x ADC</li> <li>8 x PWM Receiver Inputs</li> <li>8 Spare IO Pins</li> <li>2 x JTAG connection specifically for the TC2030-CTX-NL 6-Pin cable</li> <li>micro SD card holder</li> <li>micro USB connector</li> </ul> <p><b>Power System:</b></p> <p>New power supply based on TPS63061 DC-DC Buck-Boost</p> <p><b>Weight and Dimensions:</b></p> <p>Weight:</p> <p>Width:</p> <p>Thickness:</p> <p>Length:</p>
<b>Setup</b>	Manual
<b>Design files</b>	<p>License</p> <p>Main Board Design Files</p> <p>mIMU Board Design Files</p>

## Erle-Brain 2 autopilot

[Erle-Brain 2](#) An autopilot for making drones and robots powered by Debian/Ubuntu and with official support for the Robot Operating System (ROS). It has access to the first app store for drones and robots.

Note

[Erle-Brain 2](#) is a commercial artificial robotic brain that runs APM autopilot. It combines a Raspberry Pi 2, a sensor cape and other components in order to achieve a complete embedded Linux board.

Purchase	<a href="#">Erle-Brain2 (store)</a>
Specifications	<p><b>OS:</b> Linux Debian or Ubuntu</p> <p><b>CPU:</b> 900MHz quad-core ARM Cortex-A7 CPU</p> <p><b>Sensors:</b> Gravity sensor, gyroscope, digital compass, Pressure sensor and temperature sensor, ADC for battery sensing</p> <p><b>Interfaces:</b> 12x PWM, 1x RC IN, 1x Power Module Connector, 1x I2C connector, 1x UART connector, 4 USB ports, Full HDMI port, 10/100 Ethernet, Combined 3.5mm audio jack and composite.</p> <p><b>Camera (optional):</b> 5MP Fixed focus lens, 2592 x 1944 pixel static images, supports 1080p30, 720p60 and 640x480p60/90 video record</p> <p><b>Power System:</b> Traditional Power modules</p> <p><b>Weight and Dimensions:</b> Weight: 100 grams 70x96x20mm (without camera) 70x96x58.3mm (with camera),</p>
Setup	<a href="#">Documentation</a>

### **Erle-Brain 1 autopilot (discontinued)**

[Erle-Brain](#) An autopilot for making drones powered by Ubuntu and with official support for the Robot Operating System (ROS). It has access to the first app store for drones and robots.

#### Note

Erle-Brain is a commercial autopilot. It combines a BeagleBone Black, the [PixHawk Fire Cape](#) (above) and other components.

Purchase	<a href="#">erle-brain</a>
	<p><b>OS:</b> Linux Ubuntu</p> <p><b>CPU:</b> Cortex-A8 @ 1 GHz,</p>

<b>Specifications</b>	<p><b>Memory:</b> 512 MB DDR3 with 4GB of flash memory (8bit Embedded MMC)</p> <p><b>Sensors:</b> MPU6000, MPU9250, LSM9DS0, MS5611-01BA</p> <p><b>Interfaces:</b> SPI, 3xI2C, 2xUART, CAN, Buzzer, Safety, 8 PWM channels, PPM, S.Bus, ADC, Specktrum, 2xUSB, Ethernet</p> <p><b>Power System:</b> Traditional Power modules</p> <p><b>Weight and Dimensions:</b> Weight: 110 grams Width: 75 cm Thickness PCB: 1.6 mm Length: 92 cm</p>
<b>Setup</b>	<p><a href="#">Updating software</a>  <a href="#">BeaglePilot Project (wiki)</a>  <a href="#">Building ArduPilot for BeagleBone Black on Linux (wiki)</a></p>
<b>Design files</b>	<p><a href="#">Design files</a>  <a href="#">Erle-Brain Linux Autopilot</a></p>

## PixHawk Fire Cape (PXF)

The PixHawk Fire Cape (PXF) is a daughter board for the [BeagleBone Black](#) (BBB) development board that allows to create a fully functional Linux autopilot for drones. The combination of BBB and PXF allows to a Linux computer is a fully functional autopilot (one example is the [Erle-Brain autopilot](#)).

<b>Purchase</b>	<a href="http://erlerobotics.com/blog/product/pixhawk-fire-cape/">http://erlerobotics.com/blog/product /pixhawk-fire-cape/</a>
	<p><b>OS:</b> Linux Debian, Linux Ubuntu</p> <p><b>CPU:</b></p> <p><b>Memory:</b></p> <p><b>Sensors:</b> MPU6000, MPU9250, LSM9DS0, MS5611-01BA</p> <p><b>Interfaces:</b></p>

<b>Specifications</b>	SPI, 3xI2C, 2xUART, CAN, Buzzer, Safety, 8 PWM channels, PPM, S.Bus, ADC, Specktrum  <b>Power System:</b> Traditional Power modules
<b>Setup</b>	<a href="#">Updating the software</a>  <a href="#">BeaglePilot Project (wiki)</a>  <a href="#">Building ArduPilot for BeagleBone Black on Linux (wiki)</a>
<b>Design files</b>	<a href="https://github.com/ArduPilot/PXF">https://github.com/ArduPilot/PXF</a>

## PixHawk Fire Mini Cape (PXFmini)

The PixHawk Fire Mini Cape (PXFmini) is a daughter board designed for the low cost [Raspberry Pi zero](#) that allows to create a fully functional Linux autopilot for drones. Inspired in the PXF cape, provides a minimalist approach which allows having a reduced size/lightweight and low-cost.

<b>Purchase</b>	<a href="#">pxfmini</a>
<b>Specifications</b>	<p><b>OS:</b> Linux Debian, Linux Ubuntu</p> <p><b>CPU:</b></p> <p><b>Memory:</b></p> <p><b>Sensors:</b> MPU9250, MS5611-01BA, ADS1115</p> <p><b>Interfaces:</b></p> <p>2xI2C, 1xUART, 1xPPM-SUM, JST-GH type connectors</p> <p>8xPWM channels</p> <p><b>Power System:</b> Traditional Power modules</p> <p><b>Weight and Dimensions:</b></p> <p>Weight: 15 grams</p> <p>Dimension: 31x73mm</p>

<b>Setup</b>	<a href="#">Setup</a>
<b>Design files</b>	To be delivered in February 2016

## BBBMINI Cape

Low budget DIY Autopilot Cape for BeagleBone Black running ArduPilot on Linux.

<b>Purchase</b>	DIY
	<b>OS:</b> Debian Linux <b>CPU:</b> Cortex-A8 @ 1 GHz <b>Memory:</b> 512 MB DDR3 and 4GB of flash memory <b>Sensors:</b> MPU9250, MS5611, HC-SR04
<b>Specifications</b>	<b>Interfaces:</b> 2 x SPI, I2C, 2 x UART, CAN, 12 x PWM channels + 3 x PWM for X-Quad configuration, PPM, S.Bus, Spektrum <b>Power System:</b> Power module / UBEC <b>Weight and Dimensions:</b> Weight: 36 grams Width: 55 mm Thickness: 1.6 mm Length: 86 mm
<b>Setup</b>	<a href="https://github.com/mirkix/BBBMINI">https://github.com/mirkix/BBBMINI</a>
<b>Design files</b>	<a href="https://github.com/mirkix/BBBMINI">https://github.com/mirkix/BBBMINI</a>

## APM1 (discontinued)

An AVR2560 based autopilot with separate sensor board (aka "oilpan"). As with APM2 this is no longer supported by Copter. Not recommended for any new users.

## Closed boards

The following boards are known to be closed (they do not publish their design files).

## Parrot Bebop Drone

The [Bebop Drone](#) is a Wifi controlled quadrotor UAV that uses [this Linux autopilot](#) and which can run Copter firmware.

From Copter 3.3 the Bebop can run ArduPilot. Instructions for converting a Bebop to run ardupilot are [here](#).

<b>Purchase</b>	<a href="#">Parrot Store</a>
	<p><b>OS:</b> Linux (Busybox)</p> <p><b>CPU:</b> Parrot P7 dual-core CPU Cortex 9 with quad core GPU</p> <p><b>Memory:</b> 8GB flash</p> <p><b>Sensors:</b> MPU6050 for accelerometers and gyroscope (I2C), AKM 8963 compass, MS5607 barometer, <a href="#">Furuno GN-87F GPS</a>, Sonar, Optical-flow, HD camera</p> <p><b>Interfaces:</b> 1x UART serial ports, USB, Built-in Wifi</p> <p><b>Power System:</b></p> <p><b>Weight and Dimensions (with hull):</b> Weight: 400 grams Width: 33 cm Thickness: 38 cm Length: 36 cm</p>
<b>Specifications</b>	
<b>Setup</b>	<i>Building for Bebop on Linux &lt;<a href="#">building-for-bebop-on-linux</a>&gt; __ (wiki)</i>
<b>Design files</b>	NA

## Note

Some of this information was taken from the [Paparazzi UAV wiki page on the Bebop](#).

## NAVIO+

[NAVIO+](#) is a sensor cape for the RaspberryPi2 from Emlid. Under rapid development.

<b>Purchase</b>	<a href="http://www.emlid.com/shop/navio-plus">www.emlid.com/shop/navio-plus</a>
	<b>OS:</b>

	<p>Linux Debian</p> <p><b>CPU:</b></p> <p>?</p> <p><b>Memory:</b></p> <p>?</p> <p><b>Sensors:</b></p> <p>MPU9250 9DOF IMU</p> <p>MS5611 Barometer</p> <p>U-blox M8N Glonass/GPS/Beidou</p> <p>ADS1115 ADC for power monitoring</p> <p>MB85RC FRAM storage</p> <p>HAT EEPROM</p> <p>PCA9685 PWM generator</p> <p>RGB LED</p>
<b>Specifications</b>	<p><b>Interfaces:</b></p> <p>13 PWM servo outputs</p> <p>PPM input</p> <p>UART, SPI, I2C for extensions</p> <p><b>Power System:</b></p> <p>Triple redundant power supply</p> <p>Power module connector</p> <p><b>Weight and Dimensions:</b></p> <p>Weight: 24g</p> <p>Width: 55mm</p> <p>Thickness: ?</p> <p>Length: 65mm</p>
<b>Setup</b>	<a href="#">Emlid Documentation site</a>
<b>Design files</b>	?

## NAVIO2

[NAVIO2](#) is a new sensor cape for the RaspberryPi 3 from Emlid.

Purchase	<a href="http://www.emlid.com/shop/navio2">www.emlid.com/shop/navio2</a>
OS:	

	Linux Debian
	<b>CPU:</b>
	?
	<b>Memory:</b>
	?
	<b>Sensors:</b>
	MPU9250 9DOF IMU
	LSM9DS1 9DOF IMU
	MS5611 Barometer
	U-blox M8N Glonass/GPS/Beidou
	RC I/O co-processor
	HAT EEPROM
<b>Specifications</b>	RGB LED
	<b>Interfaces:</b>
	14 PWM servo outputs
	PPM/S.Bus input
	UART, I2C, ADC for extensions
	<b>Power System:</b>
	Triple redundant power supply
	Power module connector
	<b>Weight and Dimensions:</b>
	Weight: 23g
	Width: 55mm
	Thickness: ?
	Length: 65mm
<b>Setup</b>	<a href="#">Emlid Documentation site</a>
<b>Design files</b>	?

## VRBrain

[VRBrain](#) is a multipurpose controller board that comes loaded with a 32 bit version of Copter firmware. At time of writing the latest version is [VR Brain 5](#).

Purchase	<a href="http://vrbrain.wordpress.com/store/">vrbrain.wordpress.com/store/</a>
	<b>OS:</b>

	<p>NuttX</p> <p><b>CPU:</b></p> <p>ARM CortexM4F microcontroller with DSP and FPU.</p> <p><b>Memory:</b></p> <p>1024KB flash memory, 192KB of RAM.</p> <p><b>Sensors:</b></p> <p>mems accelerometer and gyroscope.</p> <p>barometer with 10 cm resolution.</p> <p>2 SPI expansion BUS for optional IMU</p> <p>1 sonar input.</p> <p><b>Interfaces:</b></p> <p>8 RC Input standard PPM , PPMSUM , SBUS</p> <p>8 RC Output at 490 hz</p> <p>1 integrated high speed data flash for logging data</p> <p>1 Can bus 2 i2c Bus</p> <p>3 Serial port available one for GPS 1 for serial option 1 for serial telemetry.</p> <p>3 digital switch (ULN2003).</p> <p>Jtag support for onboard realtime debugger.</p> <p>1 Buzzer output.</p> <p>1 Input for control lipo voltage</p> <p><b>Power System:</b></p> <p><b>Weight and Dimensions:</b></p> <p>Weight: ?</p> <p>Width: 4 cm</p> <p>Thickness: ?</p> <p>Length: 6 cm</p>
<b>Specifications</b>	
<b>Setup</b>	<a href="#">Quick Start Guide</a>
<b>Design files</b>	?

## Qualcomm Snapdragon Flight Kit

The [Qualcomm® Snapdragon Flight™ Kit \(Developer's Edition\)](#) is small (58x40mm) but offers a lot of CPU power and two on-board cameras. It contains 4 'Krait' ARM cores which run Linux (Ubuntu 14.04 Trusty, by default), and 3 'Hexagon' DSP cores which run the QURT RTOS. In addition it includes Wi-Fi,

Bluetooth connectivity, automotive-grade GPS and many more features.

Information about using this board with ArduPilot can be found here: [Building for Qualcomm Snapdragon Flight Kit, QURT Port](#) (Github) and [QFlight Port](#) (Github).

### Warning

Due to some rather unusual licensing terms from Intrinsyc we cannot distribute binaries of ArduPilot (or any program built with the Qualcomm libraries). So you will have to build the firmware yourself.

<b>Purchase</b>	<a href="http://shop.intrinsyc.com/products/snapdra gon-flight-dev-kit">shop.intrinsyc.com/products/snapdra gon-flight-dev-kit</a>
	<p><b>OS:</b></p> <p>Ubuntu Linux (Ubuntu 14.04 Trusty by default)</p> <p><b>System on a Chip: System-on-Chip:</b> Snapdragon 801</p> <p>CPU: Quad-core 2.26 GHz Krait</p> <p>DSP: Hexagon DSP (QDSP6 V5A) – 801 MHz+256KL2 (running the flight code)</p> <p>GPU: Qualcomm® Adreno™ 330 GPU</p> <p><b>Memory:</b></p> <p>RAM: 2GB LPDDR3 PoP @931 MHz</p> <p>Storage: 32GB eMMC Flash</p> <p><b>Sensors:</b></p> <p>GPS: Telit Jupiter SE868 V2 module</p> <p>Omnivision OV7251 on Sunny Module MD102A-200 (Optic Flow camera - 640x480)</p> <p>Sony IMX135 on Liteon Module 12P1BAD11 (4K High Res camera)</p> <p>MPU: Invensense MPU-9250 9-Axis Sensor, 3x3mm QFN</p> <p>Baro: Bosch BMP280 barometric pressure sensor</p> <p><b>Interfaces:</b></p> <p>CSR SiRFstarV @ 5Hz via UART</p> <p>uCOAX connector on-board for connection to external GPS patch antenna</p> <p>BT/WiFi: BT 4.0 and 2G/5G WiFi via QCA6234</p> <p>Wifi: Qualcomm® VIVE™ 1-stream 802.11n/ac with MU-MIMO † Integrated digital core</p> <p>802.11n, 2x2 MIMO with 2 uCOAX connectors on-board for connection to external antenna</p> <p>One USB 3.0 OTG port (micro-A/B)</p> <p>Micro SD card slot</p>
<b>Specifications</b>	

	<p>Gimbal connector (PWB/GND/BLSP)</p> <p>ESC connector (2W UART)</p> <p>I2C</p> <p>60-pin high speed Samtec QSH-030-01-L-D-A-K expansion connector</p> <p>2x BLSP (BAM Low Speed Peripheral)</p> <p>USB</p> <p><b>Power System:</b></p> <p>5VDC via external 2S-6S battery regulated down to 5V via APM adapter</p> <p><b>Weight and Dimensions:</b></p> <p>Weight: ?</p> <p>Width: 58mm for pcb (68 with pcb+connectors+camera)</p> <p>Thickness: ?</p> <p>Length: 40mm for pcb (52 with pcb+connectors+camera) Additional information can be found at <a href="http://www.intrinsyc.com/qualcomm-snapdrag-on-flight-details/">www.intrinsyc.com/qualcomm-snapdrag on-flight-details/</a> (behind short survey).</p>
<b>Setup</b>	?
<b>Design files</b>	?

## Learning the ArduPilot Codebase

### Introduction

The ArduPilot code base is quite large (about 700k lines for the core ardupilot git tree) and can be quite intimidating to a new user. This page is meant to give some suggestions on how to come up to speed on the code quickly. It assumes you already are familiar with the key concepts of C++ and the many of the examples currently assume you will be exploring the code on a Linux system.

This page and the pages linked below are designed to be used as a tutorial. You should work through each page step by step, trying things for yourself as you go. We will also be adding more pages over time. If you think some important information is missing or could be improved then please [open an issue](#) on the ArduPilot github and we'll try to get to it when we can.

### Tutorial steps

- [Introduction and code structure](#)
- [The Example Sketches](#)
- [ArduPilot threading](#)
- [UARTs and the console](#)
- [RCInput and RCOOutput](#)
- [Storage and EEPROM management](#)
- [The vehicle code](#)

## Upcoming Tutorials

We will be adding more tutorial sections in the future. Check back occasionally to see the new sections.

- MAVLink telemetry handling
- The Dataflash library for onboard logging
- Analog input
- GPIOs
- Timing and profiling
- PX4 device drivers
- I2C Drivers
- SPI Drivers
- CANBUS drivers and uavcan
- memory management
- Maths functions
- Inside the AP\_AHRS attitude and position estimator
- Porting ArduPilot to a new board
- AP\_HAL Utility functions
- PIDs and other control libraries
- Inside the SITL simulator
- Inside the AP\_Param parameter system
- AP\_Notify for buzzers and LEDs
- The ArduPilot autotest system
- How autobuilds work, and developer autobuilds
- How ardupilot boot process works on PX4

## Welcome to ArduPilot!

If you have gotten this far you should be in a good position to start getting involved with ArduPilot development. Please [join the drones-discuss mailing list](#) and join in on the discussions. We also have a weekly mumble developer conference call, some Skype chat channels and some IRC channels. Ask any ardupilot developer for details.

Welcome to ArduPilot development, we hope you enjoy being part of our community!

## Learning ArduPilot — Introduction

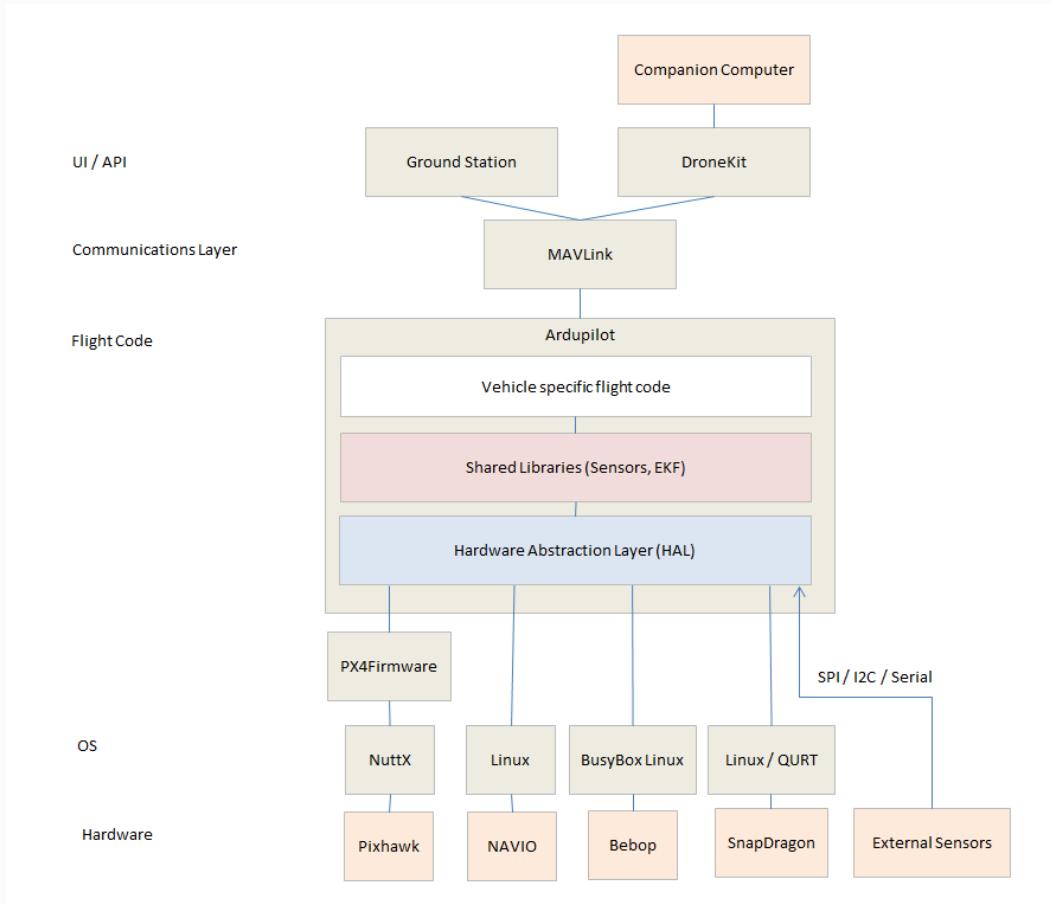
This page introduces the basic structure of ArduPilot. Before you get started you should work out what code exploring system you will use. You could just use a web browser and look at <https://github.com/ArduPilot/ardupilot/> but you will probably get a lot more out of it if you use a good programmers IDE that allows you to find function, structure and class definitions and shows the code in a structured manner.

Some suggestions are:

- Eclipse on Windows, Linux or MacOS
- Emacs on Linux, Windows or MacOS, with etags for finding code elements
- Vim on emacs with ctags

There are a lot of other IDEs available, and many of them are sufficiently customisable that they can handle something like ArduPilot nicely. If you have a favourite and have worked out how to make ArduPilot development a great experience then please consider contributing a wiki page on how to use it.

## Basic structure



The basic structure of ArduPilot is broken up into 5 main parts:

- vehicle directories
- AP\_HAL
- libraries
- tools directories
- external support code

These will be described in detail below, but before moving on make sure you have [cloned all of the git repositories](#) you will need.

### Vehicle Directories [¶](#)

The vehicle directories are the top level directories that define the firmware for each vehicle type. Currently there are 4 vehicle types - Plane, Copter, APMrover2 and AntennaTracker.

Along with the \*.cpp files, each vehicle directory contains a make.inc file which lists library dependencies. The Makefiles read this to create the -I and -L flags for the build.

### AP\_HAL [¶](#)

The AP\_HAL layer (Hardware Abstraction Layer) is how we make ArduPilot portable to lots of different platforms. There is a top level AP\_HAL in libraries/AP\_HAL that defines the interface that the rest of the code has to specific board features, then there is a AP\_HAL\_XXX subdirectory for each board type, for example AP\_HAL\_AVR for AVR based boards, AP\_HAL\_PX4 for PX4 boards and AP\_HAL\_Linux for Linux based boards.

### Tools directories [¶](#)

The tools directories are miscellaneous support directories. For examples, tools/autotest provides the autotest infrastructure behind the [autotest.ardupilot.org](http://autotest.ardupilot.org) site and tools/Replay provides our log replay utility.

## External support code

On some platforms we need external support code to provide additional features or board support.

Currently the external trees are:

- [PX4NuttX](#) - the core NuttX RTOS used on PX4 boards
- [PX4Firmware](#) - the base PX4 middleware and drivers used on PX4 boards
- [uavcan](#) - the uavcan CANBUS implementation used in ArduPilot
- [mavlink](#) - the mavlink protocol and code generator

### Note

Most of these are imported as [Git Submodules](#) when you build ArduPilot for PX4/Pixhawk.

## Build system

The build system is based around make, but also supports the old arduino IDE for AVR builds. The makefiles are in the [mk/ directory](#), and define build rules for each type of supported board

To build a vehicle or other ‘sketch’ for a particular board target you would type “make TARGET”, where TARGET is the board type. The following board types are currently available:

- make apm1 - the APM1 board
- make apm2 - the APM2 board
- make px4-v1 - the PX4v1
- make px4-v2 - the Pixhawk (and [Arsov AUAV-X2](#))
- make pxf - the BBB+PXF cape combination
- make navio - the RaspberryPi+NavIO cape combination
- make linux - a generic Linux build
- make flymaple - the FlyMaple board
- make vrbain - the VRBrain boards
- make sitl - the SITL software in the loop simulation

More ports are being added all the time, so check “make help” file for new targets.

For each of these builds you can add additional qualifiers, and on some you can do a parallel build to speed things up. For example, in the Copter directory you could do:

```
make apm2-octa -j8
```

meaning do a build for OctaCopter on apm2 with an 8 way parallel build. You should also look into enabling [ccache](#) for faster builds.

Some boards also support upload of firmware directly from make. For example:

```
make px4-v2-upload
```

will build and upload a sketch on a Pixhawk.

There are also helper make targets for specific boards, such as:

- make clean - clean the build for non-px4 targets
- make px4-clean - completely clean the build for PX4 targets
- make px4-cleandep - cleanup just dependencies for PX4 targets

## Learning ArduPilot — The Example Sketches

The first step in exploring the code for yourself is to use the example sketches for the libraries. Following the arduino tradition we have example sketches for most libraries. A 'sketch' is just a main program, written as a cpp file.

Knowing the library API and conventions used in ArduPilot is essential to understanding the code. So using the library example sketches is a great way to get started. As a start you should read, build and run the example sketches for the following libraries:

- libraries/AP\_GPS/examples/GPS\_AUTO\_test
- libraries/AP\_InertialSensor/examples/INS\_generic
- libraries/AP\_Compass/examples/AP\_Compass\_test
- libraries/AP\_Baro/examples/BARO\_generic
- libraries/AP\_AHRS/examples/AHRS\_Test

For example, the following will build and install the AP\_GPS example sketch on a Pixhawk:

```
cd $ARDUPILOT_HOME # the top-level of an ArduPilot repository
./waf configure --board=px4-v2
./waf build --target examples/INS_generic --upload
```

waf can list the examples it can build:

```
cd $ARDUPILOT_HOME
./waf list | grep 'examples'
```

Once you have uploaded the example you can look at the output by attaching to the console. What the console is depends on the type of board. On PX4 boards (ie. PX4v1 and Pixhawk) it is the USB connector. So just connect to the USB device with your favourite serial program (the baudrate doesn't matter).

For example, if you have mavproxy installed, you could do this to connect to a Pixhawk on Linux:

```
mavproxy.py --setup --master /dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v2.x_0-if00
```

Using the `--setup` option puts mavproxy into raw serial mode, instead of processed MAVLink mode. That is what you need for the example sketches.

### Understanding the example sketch code

When you are reading the example sketch code (such as the `GPS_AUTO_test` code) you will notice a few things that may seem strange at first:

- it declares a 'hal' variable as a reference
- the code is quite rough and not well commented
- the `setup()` and `loop()` functions

The hal reference [¶](#)

Every file that uses features of the AP\_HAL needs to declare a hal reference. That gives access to a `AP_HAL::HAL` object, which provides access to all hardware specific functions, including things like printing messages to the console, sleeping and talking to I2C and SPI buses.

The actual hal variable is buried inside the board specific `AP_HAL_XXX` libraries. The reference in each file just provides a convenient way to get at the hal.

The most commonly used hal functions are:

- hal.console->printf() to print strings
- AP\_HAL::millis() and AP\_HAL::micros() to get the time since boot
- hal.scheduler->delay() and hal.scheduler->delay\_microseconds() to sleep for a short time
- hal gpio->pinMode(), hal gpio->read() and hal gpio->write() for accessing GPIO pins
- I2C access via hal.i2c
- SPI access via hal.spi

Go and have a look in the libraries/AP\_HAL directory now for the full list of functions available on the HAL.

#### [The setup\(\) and loop\(\) functions](#)

You will notice that every sketch has a setup() function and loop() function. The setup() function is called when the board boots. The actual call comes from within the HAL for each board, so the main() function is buried inside the HAL, which then calls setup() after board specific startup is complete.

The setup() function is only called once, and typically initialises the libraries, and maybe prints a “hello” banner to show it is starting.

After setup() is finished the loop() function is continuously called (by the main code in the AP\_HAL). The main work of the sketch is typically in the loop() function.

Note that this setup()/loop() arrangement is only the tip of the iceberg for more complex boards. It may make it seem that ArduPilot is single threaded, but in fact there is a lot more going on underneath, and on boards that have threading (such as PX4 and Linux based boards) there will in fact be lots of realtime threads started. See the section on understanding ArduPilot threading below.

#### [The AP\\_HAL\\_MAIN\(\) macro](#)

You will notice a extra line like this at the bottom of every sketch:

```
AP_HAL_MAIN();
```

That is a HAL macro that produces the necessary code to declare a C++ main function, along with any board level initialization code for the HAL. You rarely have to worry about how it works, but if you are curious you can look for the #define in the AP\_HAL\_XXX directories in each HAL. It is usually in AP\_HAL\_XXX\_Main.h.

#### [Rough Example code](#)

You will notice that the example sketches are quite rough, and badly commented. This is your opportunity to make a contribution to the code! As you read through the example sketches and explore how they work add some comments to the code to explain the APIs and then [submit a pull request](#) so others can benefit from your study.

## **Learning ArduPilot - Threading**

### **Understanding ArduPilot threading**

Once you have learned the basic of the ArduPilot libraries it is time for you to understand how ArduPilot deals with threading. The setup()/loop() structure that was inherited from arduino may make it seem that ArduPilot is a single threaded system, but in fact it isn't.

The threading approach in ArduPilot depends on the board it is built for. Some boards (such as the APM1 and APM2) don't support threads, so make do with a simple timer and callbacks. Some boards (PX4 and

Linux) support a rich Posix threading model with realtime priorities, and these are used extensively by ArduPilot.

There are a number of key concepts related to threading that you need to understand in ArduPilot:

- The timer callbacks
- HAL specific threads
- driver specific threads
- arduPilot drivers versus platform drivers
- platform specific threads and tasks
- the AP\_Scheduler system
- semaphores
- lockless data structures

#### The timer callbacks¶

Every platform provides a 1kHz timer in the AP\_HAL. Any code in ArduPilot can register a timer function which is then called at 1kHz. All registered timer functions are called sequentially. This very primitive mechanism is used as it is extremely portable, and yet very useful. You register a timer callback by calling the hal.scheduler->register\_timer\_process() like this:

```
hal.scheduler->register_timer_process(AP_HAL_MEMBERPROC(&AP_Baro_MS5611::_update));
```

That particular example is from the MS5611 barometer driver. The AP\_HAL\_MEMBERPROC() macro provides a way to encapsulate a C++ member function as a callback argument (bundling up the object context with the function pointer).

When a piece of code wants something to happen at less than 1kHz then it should maintain its own “last\_called” variable and return immediately if not enough time has passed. You can use the hal.scheduler->millis() and hal.scheduler->micros() functions to get the time since boot in milliseconds and microseconds to support this.

You should now go and modify an existing example sketch (or create a new one) and add a timer callback. Make the timer increment a counter then print the value of the counter every second in the loop() function. Modify your function so that it increments the counter every 25 milliseconds.

#### HAL specific threads¶

On platforms that support real threads the AP\_HAL for that platform will create a number of threads to support basic operations. For example, on PX4 the following HAL specific threads are created:

- The UART thread, for reading and writing UARTs (and USB)
- The timer thread, which supports the 1kHz timer functionality described above
- The IO thread, which supports writing to the microSD card, EEPROM and FRAM

Have a look in Scheduler.cpp inside each AP\_HAL implementation to see what threads are created and what the realtime priority of each thread is.

If you have a Pixhawk then you should also now setup a debug console cable and attach to the nsh console (the serial5 port). Connect at 57600. When you have connected, try the “ps” command ad you will get something like this:

PID	PRI	SCHD	TYPE	NP	STATE	NAME
0	0	FIFO	TASK	READY	Idle	Task()
1	192	FIFO	KTHREAD	WAITSIG	hpwork()	
2	50	FIFO	KTHREAD	WAITSIG	lpwork()	
3	100	FIFO	TASK	RUNNING	init()	
37	180	FIFO	TASK	WAITSEM	AHRS_Test()	
38	181	FIFO	PTHREAD	WAITSEM	<pthread>(20005400)	
39	60	FIFO	PTHREAD	READY	<pthread>(20005400)	

```

40 59 FIFO PTHREAD_WAITSEM <pthread>(20005400)
10 240 FIFO TASK_WAITSEM px4io()
13 100 FIFO TASK_WAITSEM fmuservo()
30 240 FIFO TASK_WAITSEM uavcan()

```

In this example you can see the “AHRS\_Test” thread, which is running the example sketch from libraries/AP\_AHRS/examples/AHRS\_Test. You can also see the timer thread (priority 181), the UART thread (priority 60) and the IO thread (priority 59).

Additionally you can see the px4io, fmuservo, uavcan, lpwork, hpwork and idle tasks. More about those later.

Other AP\_HAL ports have more or less threads depending on what is needed.

One common use of threads is to provide drivers a way to schedule slow tasks without interrupting the main autopilot flight code. For example, the AP\_Terrain library needs to be able to do file IO to the microSD card (to store and retrieve terrain data). The way it does this is it calls the function hal.scheduler->register\_io\_process() like this:

```
hal.scheduler->register_io_process(AP_HAL_MEMBERPROC(&AP_Terrain::io_timer));
```

The sets up the AP\_Terrain::io\_timer function to be called regularly. That is called within the boards IO thread, meaning it is a low realtime priority and is suitable for storage IO tasks. It is important that slow IO tasks like this not be called on the timer thread as they would cause delays in the more important processing of high speed sensor data.

#### Driver specific threads [¶](#)

It is also possible to create driver specific threads, to support asynchronous processing in a manner specific to one driver. Currently you can only create driver specific threads in a manner that is platform dependent, so this is only appropriate if your driver is intended to run only on one type of autopilot board. If you want it to run on multiple AP\_HAL targets then you have two choices:

- you can use the register\_io\_process() and register\_timer\_process() scheduler calls to use the existing timer or IO threads
- you can add a new HAL interface that provides a generic way to create threads on multiple AP\_HAL targets (please contribute patches back)

An example of a driver specific thread is the ToneAlarm thread in the Linux port. See AP\_HAL\_Linux/ToneAlarmDriver.cpp

#### ArduPilot drivers versus platform drivers [¶](#)

You may notice some driver duplication in ArduPilot. For example, we have a MPU6000 driver in libraries/AP\_InertialSensor/AP\_InertialSensor\_MP6000.cpp, and another MPU6000 driver in PX4Firmware/src/drivers/mpu6000.

The reason for this duplication is that the PX4 project already provides a set of well tested drivers for hardware that comes with PX4 boards, and we enjoy a good collaborative relationship with the PX4 project on developing and enhancing these drivers. So when we build ArduPilot for PX4 we take advantage of the PX4 drivers by writing small “shim” drivers which present the PX4 drivers with the standard ArduPilot library interface. If you look at libraries/AP\_InertialSensor/AP\_InertialSensor\_PX4.cpp you will see a small shim driver that asks the PX4 what IMU drivers are available on this board and automatically makes all of them available as part of the ArduPilot AP\_InertialSensor library.

So if we have an MPU6000 on the board we use the AP\_InertialSensor\_MP6000.cpp driver on non-PX4 platforms, and the AP\_InertialSensor\_PX4.cpp driver on PX4 based platforms.

The same type of split can also happen for other AP\_HAL ports. For example, we could use Linux kernel

drivers for some sensors on Linux boards. For other sensors we use the generic AP\_HAL I2C and SPI interfaces to use the ArduPilot “in-tree” drivers which work across a wide range of boards.

#### Platform specific threads and tasks¶

On some platforms there will be a number of base tasks and threads that will be created by the startup process. These are very platform specific so for the sake of this tutorial I will concentrate on the tasks used on PX4 based boards.

In the “ps” output above we saw a number of tasks and threads that were not started by the AP\_HAL\_PX4 Scheduler code. Specifically they are:

- idle task - called when there is nothing else to run
- init - used to start up the system
- px4io - handle the communication with the PX4IO co-processor
- hwork - handle thread based PX4 drivers (mainly I2C drivers)
- lpwork - handle thread based low priority work (eg. IO)
- fmuservo - handle talking to the auxillary PWM outputs on the FMU
- uavcan - handle the uavcan CANBUS protocol

The startup of all of these tasks is controled by the PX4 specific [rc.APM script](#). That script is run when the PX4 boots, and is responsible for detecting what sort of PX4 board we are using then loading the right tasks and drivers for that board. It is a “nsh” script, which is similar to a bourne shell script (though nsh is much more primitive).

As an exercise, try editing the rc.APM script and adding some sleep and echo commands. Then upload a new firmware and connect to the debug console while the board is booting. Your echo commands should show up on the console.

Another very useful way of exploring the startup of the PX4 is to boot without a microSD card in the slot. The [rcS script](#), which runs just before rc.APM, detects if a microSD is inserted and gives you a bare nsh console on the USB port if it isn’t. You can then manually run all the steps of rc.APM yourself on the USB console to learn how it works.

Try the following exercise after booting a Pixhawk without a microSD card and connecting to the USB console:

```
tone_alarm stop
uorb start
mpu6000 start
mpu6000 info
mpu6000 test
mount -t binfs /dev/null /bin
ls /bin
perf
```

Try playing with the other drivers. Have a look in /bin to see what is available. The source code for most of these commands is in [PX4Firmware/src/drivers](#). Have a look through the mpu6000 driver to get an idea of what is involved.

Given we are on the topic of threads and tasks, a brief description of threads in the PX4Firmware git tree is worth mentioning. If you look in the mpu6000 driver you will see a line like this:

```
hrt_call_every(&_call, 1000, _call_interval, (hrt_callout)&MPU6000::measure_trampoline, this);
```

that is the equivalent of the hal.scheduler->register\_timer\_process() function in the AP\_HAL, but is PX4 specific and is also much more flexible. It says that it wants the HRT (high resolution timer) subsystem of the PX4 to call the MPU6000::measure\_trampoline function every 1000 microseconds.

Using `hrt_call_every()` is the common method used for regular events in drivers where the operations are very fast, such as SPI device drivers. The operations are typically run with interrupts disabled, and should take only a few tens of microseconds at most.

If you compare this to the hmc5883 driver, you will instead see a line like this:

```
work_queue(HWORK, &work, (worker_t)&HMC5883::cycle_trampoline, this, 1);
```

that uses an alternative mechanism for regular events which is suitable for slower devices, such as I2C devices. What this does is add the `cycle_trampoline` function to a work queue within the hwork thread that you saw above. Calls made within HWORK workers should run with interrupts enabled and may take up to a few hundred microseconds. For tasks which will take longer than that the LPWORK work queue should be used, which runs them in the lower priority lpwork thread.

### The AP\_Scheduler system¶

The next aspect of ArduPilot threading and tasks to understand is the AP\_Scheduler system. The AP\_Scheduler library is used to divide up time within the main vehicle thread, while providing some simple mechanisms to control how much time is used for each operation (called a ‘task’ in AP\_Scheduler).

The way it works is that the `loop()` function for each vehicle implementation contains some code that does this:

- wait for a new IMU sample to arrive
- call a set of tasks between each IMU sample

It is a table driven scheduler, and each vehicle type has a `AP_Scheduler::Task` table. To learn how it works have a look at the [AP\\_Scheduler/examples/Scheduler\\_test.cpp](#) sketch.

If you look inside that file you will see a small table with a set of 3 scheduling tasks. Associated with each task are two numbers. The table looks like this:

```
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
{ ins_update, 1, 1000 },
{ one_hz_print, 50, 1000 },
{ five_second_call, 250, 1800 },
};
```

The first number after each function name is the call frequency, in units controlled by the `ins.init()` call. For this example sketch the `ins.init()` uses RATE\_50HZ, so each scheduling step is 20ms. That means the `ins_update()` call is made every 20ms, the `one_hz_print()` function is called every 50 times (ie. once a second) and the `five_second_call()` is called every 250 times (ie. once every 5 seconds).

The third number is the maximum time that the function is expected to take. This is used to avoid making the call unless there is enough time left in this scheduling run to run the function. When `scheduler.run()` is called it is passed the amount of time (in microseconds) available for running tasks, and if the worst case time for this task would mean it wouldn’t fit before that time runs out then it won’t be called.

Another point to look at closely is the `ins.wait_for_sample()` call. That is the “metronome” that drives the scheduling in ArduPilot. It blocks execution of the main vehicle thread until a new IMU sample is available. The time between IMU samples is controlled by the arguments to the `ins.init()` call.

Note that tasks in AP\_Scheduler tables must have the following attributes:

- they should never block (except for the `ins.update()` call)
- they should never call sleep functions when flying (an autopilot, like a real pilot, should never sleep

- while flying)
- they should have predictable worst case timing

You should now go and modify the Scheduler\_test example and add in your own tasks to run. Try adding tasks that do the following:

- read the barometer
- read the compass
- read the GPS
- update the AHRS and print the roll/pitch

Look at the example sketches for each library that you worked with earlier in this tutorial to understand how to use each sensor library.

### Semaphores

When you have multiple threads (or timer callbacks) you need to ensure that data structures shared by the two logical threads of execution are updated in a way that prevents corruption. There are 3 principle ways of doing this in ArduPilot - semaphores, lockless data structures and the PX4 ORB.

AP\_HAL Semaphores are just wrappers around whatever semaphore system is available on the specific platform, and provide a simple mechanism for mutual exclusion. For example, I2C drivers can ask for the I2C bus semaphore to ensure that only one I2C device is used at a time.

Go and have a look at the hmc5843 driver in libraries/AP\_Compass/AP\_Compass\_HMC5843.cpp and look for the get\_semaphore() call. Look at all the places it is used, and see if you can work out why it is needed.

### Lockless Data Structures

The ArduPilot code also contains examples of using lockless data structures to avoid the need for a semaphore. This can be a lot more efficient than semaphores.

Two examples of lockless data structures in ArduPilot are:

- the \_shared\_data structure in libraries/AP\_InertialSensor/AP\_InertialSensor\_MP9250.cpp
- the ring buffers used in numerous places. A good example is libraries/DataFlash/DataFlash\_File.cpp

Go and have a look at these two examples, and prove to yourself that they are safe for concurrent access. For DataFlash\_File look at the use of the \_writebuf\_head and \_writebuf\_tail variables.

It would be nice to create a generic ring buffer class which could be used instead of the separate ringbuffer implementations in several places in ArduPilot. If you want to contribute that then please do a pull request!

### The PX4 ORB

Another example of this type of mechanism is the PX4 ORB. The ORB (Object Request Broker) is a way of providing data from one part of the system to another (eg. device driver -> vehicle code) using a publish/subscribe model that is safe in a multi-threaded environment.

The ORB provides a nice mechanism for declaring structures which will be shared in this way (all defined in [PX4Firmware/src/modules/uORB/topics](#)). Code can then “publish” data to one of these topics, which is picked up by other pieces of code.

An example is the publication of actuator values so the uavcan ESCs can be used on Pixhawk. Have a look at the \_publish\_actuators() function in AP\_HAL\_PX4/RCOutput.cpp. You will see that it advertises a “actuator\_direct” topic, which contains the speed desired for each ESC. The uavcan code then watches for changes to this topic in [PX4Firmware/src/modules/uavcan/uavcan\\_main.cpp](#) and outputs the new values to the uavcan ESCs.

Two other common mechanisms for communicating with PX4 drivers are:

- ioctl calls (see the examples in AP\_HAL\_PX4/RCOutput.cpp)
- /dev/xxx read/write calls (see \_timer\_tick in AP\_HAL\_PX4/RCOutput.cpp)

Please talk to the ardupilot development team on the drones-discuss mailing list if you are not sure which mechanism to use for new code.

## Learning ArduPilot - UARTs and the Console

A lot of components in ArduPilot rely on UARTs. They are used for debug output, telemetry, GPS modules and more. Understanding how to talk to the UARTs via the HAL will help you understand a lot of ArduPilot code.

### The 5 UARTs

The ArduPilot HAL currently defines 5 UARTs. The HAL itself doesn't define any particular roles for these UARTs, but the other parts of ArduPilot assume they will be assigned particular functions

- uartA - the console (usually USB, runs MAVLink telemetry)
- uartB - the first GPS
- uartC - primary telemetry (telem1 on Pixhawk, 2nd radio on APM2)
- uartD - secondary telemetry (telem2 on Pixhawk)
- uartE - 2nd GPS

If you are writing your own sketch using the ArduPilot HAL then you can use these UARTs for any purpose you like, but if possible you should try to use the above assignments as it will allow you to fit in more easily to existing code.

Some UARTs have dual roles. For example there is a parameter SERIAL2\_PROTOCOL changes uartD from being used for MAVLink versus being used for Frsky telemetry.

Go and have a look at the [libraries/AP\\_HAL/examples/UART\\_test](#) example sketch. It prints a hello message to all 5 UARTs. Try it on your board and see if you can get all the outputs displaying using a USB serial adapter. Try changing the baudrate in the sketch.

Debug console¶

In addition to these 5 UARTs there is an additional debug console available on some platforms. You can tell if the platform you are on has a debug console by checking for the HAL\_OS\_POSIX\_IO macro, like this:

```
#if HAL_OS_POSIX_IO
    ::printf("hello console\n");
#endif
```

If you have a board that does have HAL\_OS\_POSIX\_IO set (check that in [AP\\_HAL/AP\\_HAL\\_Blocks.h](#)) then try adding some ::printf() and other stdio functions to the UART\_test sketch.

Note that on some boards (eg. the Pixhawk) hal.console->printf() goes to a different place to ::printf(). On the Pixhawk a hal.console->printf() goes to the USB port whereas ::printf() goes to the dedicated debug console (which also runs the nsh shell for NuttX access).

### UART Functions

Every UART has a number of basis IO functions available. The key functions are:

- printf - formatted print
- printf\_P - formatted print with PROGMEM string (saves memory on AVR boards)
- println - print and line feed
- write - write a bunch of bytes
- read - read some bytes
- available - check if any bytes are waiting
- txspace - check how much outgoing buffer space is available
- get\_flow\_control - check if the UART has flow control capabilities

Go and have a look at the declarations of each of these in AP\_HAL and try them in UART\_test.

## **Learning ArduPilot — RC Input and Output**

RC input is a key part of any autopilot, giving the pilot control of the airframe, allowing them to change modes and also giving them control of auxiliary equipment such as camera mounts.

ArduPilot supports several different types of RC Input depending on the board type:

- PPMSum - on PX4, Pixhawk, Linux and APM2
- SBUS - on PX4, Pixhawk and Linux
- Spektrum/DSM - on PX4, Pixhawk and Linux
- PWM - on APM1 and APM2
- RC Override (MAVLINK) - all boards

The number of channels available depends on the hardware of the particular board. Note that SBUS and Spektrum/DSM are serial protocols. SBUS is a 100kbaud inverted UART protocol and Spektrum/DSM is a 115200 baud UART protocol. Some boards implement these using hardware UARTs (such as on PX4) and some implement them as bit-banged software UARTs (on Linux).

RC Output is how ArduPilot controls servos and motors. The number of available output channels depends on the type of board, and can even depend on the vehicle type and configuration parameters. RC Output defaults to 50Hz PWM values, but can be configured for a wide range of update rates. For example, the Copter code sets up its motor outputs to drive the ESCs at a much higher rate - typically over 400Hz.

### **AP\_HAL RCInput object**

The first object to understand is the AP\_HAL RCInput object which is available as hal.rcin. That provides low level access to the channel values currently being received on the board. The returned values are PWM values in microseconds.

Go and have a look at the [libraries/AP\\_HAL/examples/RCInput/RCInput.cpp](#) sketch and try it on your board. Try moving the sticks on your transmitter and check that the values change correctly in the output.

### **AP\_HAL RCOOutput object**

The AP\_HAL RCOOutput object (available as hal.rcout) gives low level control of all output channels. How this is implemented is very board specific, and may involve programming of on-chip timers, or an I2C peripheral, or output via a co-processor (such as the PX4IO microcontroller).

Go and have a look at the [libraries/AP\\_HAL/examples/RCOutput/RCOutput.cpp](#) example sketch. You'll see that it just sets up all the channels to wave the servos from minimum to maximum over a few second period. Hook up some servos to your board and then test to make sure it works for you.

## The RC\_Channel object

The hal.rcin and hal.rcout objects discussed above are low level functions. The usual way of handling RC input and output in ArduPilot is via a higher level object called RC\_Channel. That object has user configurable parameters for the min/max/trim of each channel, as well as support for auxillary channel function, scaling of inputs and outputs and many other features.

Go and have a look at [libraries/RC\\_Channel/examples/RC\\_Channel/RC\\_Channel.cpp](#). That example shows how to setup RC channels, read input and copy input to output values. Run that on your board and check that transmitter input is passed through to a servo. Try changing it to reverse a channel, and change the min/max/trim on a channel. Have a look through RC\_Channel.h to see what API functions are available.

The strange input/output setup in RC\_Channel¶

If you look at RC\_Channel carefully you'll notice something strange. A lot of variables apply to both the input side and the output side. For example, the rc1->radio\_trim applies to channel 1 as an input (specifying the trim point for calculation a input proportion), plus as an output, specifying the midpoint of a servo attached to that channel.

This ties input channels numbers to output channel numbers, when really they are separate concepts. You may want to use channel 1 input as "roll input", but use channel 1 output for steering a nose wheel on an aircraft. With RC\_Channel you can't really do that, or at least if you do it you will end up with some very odd code. This is an artifact of how RC\_Channel was originally developed, where pass-thru from input channels to output channels on a fixed wing aircraft in manual mode was normal. Someday we may change it to break apart the two concepts in a more logical fashion, but for now just be aware that it is strange, so when you see odd bits of code working around this you'll know why.

## The RC\_Channel\_aux object

Along with RC\_Channel there is another important class in libraries/RC\_Channel. It is the RC\_Channel\_aux class, which is a subclass of RC\_Channel.

An RC\_Channel\_aux object is a type of RC\_Channel but with additional properties that allow its purpose to be specified by a user. Say for example a user wants channel 6 to be a roll stabilization for a camera mount. They would set a parameter RC6\_FUNCTION to be 21, which means "rudder". Then another library could say:

```
RC_Channel_aux::set_servo_out(RC_Channel_aux::k_rudder, 4500);
```

and any channel which has its FUNCTION set to 21 will move to full deflection (as k\_rudder is setup scaled as an angle in centi-degrees with 4500 being the maximum). Note that this is a one to many arrangement. The user can setup multiple channels to have FUNCTION 21 if they want to, and all of those channels will move, each using their own min/max/trim values.

## Learning ArduPilot - Storage and EEPROM management

Every board that ArduPilot supports has some form of persistent storage available. This is used to hold user parameters, waypoints, rally points, terrain data and many other useful things. To provide access to this storage ArduPilot has 4 basic mechanisms:

- the AP\_HAL::Storage object, accessed as hal.storage
- the StorageManager library to give a higher level abstraction layer on hal.storage
- DataFlash for storing to an on-board logging area

- Posix IO functions to traditional filesystems (for example VFAT on a microSD card), on boards that support it

The other libraries and functions that need persistent storage are built on these basic systems. For example, the AP\_Param library (which handles user settable parameters) is built on top of StorageManager, which in turn is built on top of AP\_HAL::Storage. The AP\_Terrain library (which handles terrain data) is built on top of Posix IO functions for holding the terrain database.

### **The AP\_HAL::Storage library**

The AP\_HAL::Storage object is available on all platforms. The minimum size of storage available via this interface on boards that ArduPilot supports is 4096 bytes. Some boards provide more space - for example the PX4v1 has 8k of EEPROM and the Pixhawk has 16k of FRAM. All of that is hidden behind the AP\_HAL::Storage API.

The hal.storage API is extremely simple. It has just 3 functions

- init() to start up the storage subsystem
- read\_block() to read a block of bytes
- write\_block() to write a block of bytes

The reason for this very simple API is that developers are encouraged to use the StorageManager API instead of hal.storage. You should only be delving into hal.storage when doing bringup of a new board, or when debugging.

The size of storage available is set inside [AP\\_HAL/AP\\_HAL\\_Blocks.h](#) in the macro HAL\_STORAGE\_SIZE. This means we don't yet support dynamically determining the size of storage available for this interface. If you want dynamically sized storage you need to use Posix IO for now.

We don't have an example sketch for the AP\_HAL::Storage API, so this is your chance to write one. If you have gotten this far into the ArduPilot tutorial you should have seen enough example sketches to know how to write one from scratch. So write a [libraries/AP\\_HAL/examples/Storage](#) example that calculates an 8 bit XOR of the full contents of the hal.storage data, and prints it to the console. Then [submit the example as a patch](#) to the ArduPilot github, being careful to follow the patch submission guidelines.

I will be quite interested to see how long after I add this exercise to the tutorial before we get a submission ....

### **The StorageManager library**

The simple hal.storage API makes it nice and easy to port ArduPilot to a new board, but isn't convenient for actually using the available storage. For that we have StorageManager. The StorageManager library provides an API which abstracts the storage into pseudo-contiguous blocks of data that have an assigned purpose. So we chop up the available storage into areas that provide:

- parameters
- fence points
- waypoints
- rally points

Go and have a read of [libraries/StorageManager/StorageManager.cpp](#). In particular look at the tables at the top. Notice how multiple areas of each type are defined for systems with have larger amounts of storage. This ability to combine multiple non-contiguous areas of storage into a single logical storage area was the main motivation for adding StorageManager. It allowed us to grow from using 4k of storage on all boards to using the full storage available on each board without asking users to reload all their parameters or reload their waypoints.

This theme of trying to avoid making users reconfigure their autopilot boards is a common one in ArduPilot. We like it when users can update to the latest firmware and everything they previously setup still works, while gaining new features. Sometimes that means we have to do a bit more work as developers to make that possible - such as happened with StorageManager.

The StorageManager API also provides convenient functions for reading and writing variables like integers. This is the API that libraries like AP\_Mission use to save and restore waypoints.

Now go and have a look at [libraries/StorageManager/examples/StorageTest.cpp](#). This is a stress test for the StoageManager layer, and as a result is also a stress test for the AP\_HAL::Storage object. It does random IO at random offsets of random lengths. That means it does IOs that cross boundaries where a single parameter value may be non-contiguous in FRAM or EEPROM. This test sketch was designed to stress test the StorageManager API, but it is also terribly useful when porting ArduPilot to a new board, as it stresses the EEPROM access functions very nicely.

Go and try StorageTest on your board, but be warned that it is a destructive test. It won't destroy your board, but it will wipe all your parameters and waypoints. So backup your configuration if you are testing on the board in your favourite aircraft.

As an exercise, add some profiling to StorageTest so it prints the total IO rate in kbytes/sec along with the IO rate for reads and writes. Do you notice something about the difference between the read and write rates? Do you notice anything about the speed of writing values which are already set at that address in storage? See if you can find the code in StorageManager that explains your observations. Then [submit a patch](#) adding the profiling output to the ArduPilot github.

Next go and search through the ArduPilot libraries to fine all the places that use the StorageManager. What you are looking for is StorageAccess handles, like this:

```
StorageAccess AP_Param::_storage(StorageManager::StorageParam);
```

that declares a variable called AP\_Param::\_storage which provides access to the StorageParam area of the storage on this board. What libraries use StorageAccess?

### **The DataFlash library**

Another type of storage that autopilots need is for on-board logs. For ArduPilot that is provided by the DataFlash library. It is quite an odd library in many ways. For a start the name is weird and comes from the fact that it was originally designed around the DataFlash chip for the APM1. It was a hardware device driver library which morphed over time into a general logging system. Internally the structure of the library shows this history in several ways (and not good ways!).

These days the DataFlash API is designed around a logging infrastructure model. It allows you to define self-describing data structures for individual log messages - such as a "GPS" message to log data from a GPS sensor. It handles logging that data to persistent storage in an efficient manner and also provides APIs for other libraries to use to get the data back out when the user wants to download their log files after a flight.

If you have seen the \*.bin' files that ArduPilot uses these days when you download a log then you have seen the format that ArduPilot uses to store log messages. It is "self describing", meaning that the ground station can work out the format of the messages in the log file without having to have some common scheme. At the front of each log file is a set of FMT messages which have a well known format and which describe the format of the messages that follow.

Go and have a look at [libraries/DataFlash/examples/DataFlash\\_test/DataFlash\\_test.cpp](#). You'll see a little table at the top that defines the log messages we will be writing, in this case a 'TEST' message which contains 4 unsigned 16 bit integers and two signed 32 bit integers (that is what "HHHii" means). It also

gives names for those 6 variables (cunningly labelled V1 to V4 and L1 and L2).

In the loop() function you will see a rather strange call like this:

```
DataFlash.get_log_boundaries(log_num, start, end);
```

This is the public API for the way that the DataFlash library hides how the board actually stores log files. On a system that has Posix IO (such as Pixhawk or Linux) log files are stored as separate files in a "LOGS" directory on the microSD card. These files can be directly copied by a user by pulling out the microSD card and putting it into their PC.

On a board like the APM2 things aren't quite that simple. The APM2 has 4 megabytes of storage on a DataFlash chip, accessible across an SPI interface. The interface itself is page oriented, so you need to fill one 512 byte (or possibly 528 byte!) page, then tell the chip to copy that page to persistent storage while you fill the next page. Doing random IO on this DataFlash is not good - it is designed for use by code that needs to write continuously, which is what happens when logging. The 4 megabyte size is really not very large compared to the amount of data an autopilot likes to log, so we need to handle wrapping when it fills up as well.

All of that complexity is hidden behind an API that presents the concept of a "log number", which is just a bunch of bytes that were written in one flight of the autopilot. The DataFlash implementation on APM1 and APM2 uses little marker bytes at the front of each page to say which log number is being written. These log numbers correspond to the log numbers that are downloaded when the user asks to retrieve their logs.

## Posix IO

Some of the autopilot boards that ArduPilot supports are based on an operating system that has a Posix-like API. For example the Linux ports have a very good Posix subsystems, and the NuttX operating system used for PX4 (such as on Pixhawk) have a pretty reasonable Posix layer. You can take advantage of this in libraries for ArduPilot as long as you don't rely on it for anything that has to work on all boards.

A good example of this is the AP\_Terrain library, which holds terrain data. Terrain data is much too large to fit in EEPROM, and it is random access, so it isn't suitable for DataFlash. It is also not essential for the basic functionality of an autopilot, so it is a great candidate for implementing using Posix IO.

The way we use Posix IO is that you first check if the board has Posix IO support by checking the HAVE\_OS\_POSIX\_IO macro from [AP\\_HAL\\_Blocks.h](#). Then to know where on the filesystem you should store the data you add a data specific macro in AP\_HAL\_Blocks.h which gives the directory path where that sort of data should be placed. For example, the macro HAL\_BOARD\_TERRAIN\_DIRECTORY is used to define the directory where terrain data should go.

Once you have those two things you should just use normal Posix IO functions (ie. open, close, read, write etc), although there are some caveats:

- you must only call IO functions from either the IO timer or from your own low priority thread.
- Never call any IO functions from APIs directly accessible in your library. Not even a simple one like stat().
- try to be friendly to slow storage cards, do IO in reasonably sized chunks, and avoid seeks where possible

These rules really matter. A simple IO on a microSD card on Pixhawk could take up to a second. That is enough time for your precious quadcopter to be flying upside down and heading for its final meeting with the ground. The average time for an IO on a Pixhawk microSD card is quite low (a few milliseconds), but just occasionally you will get a slow one when the microSD card decides it needs to spend some quality time re-reading the SD card specification and calculating Pi. Don't be tempted to sneak a little operation

in just because it seems fast most of the time.

The one exception to this is initialization functions that you know for certain can only be called when the vehicle is starting up or is disarmed. A bit of delay at that time is fine.

Now go and have a read of [libraries/AP\\_Terrain/TerrainIO.cpp](#) and look at how it uses Posix IO. Notice the little state machine it uses to handle all the IO, which is all called from the `AP_Terrain::io_timer` function. See if you can spot any bugs, and report them if you can!

## Learning ArduPilot - Vehicle Code

Now that you understand the ArduPilot libraries and the way tasks work in ArduPilot it is time to start exploring a particular vehicle type. There are currently 4 vehicles in ArduPilot:

- Copter - for multicopters and helitopters
- Plane - for fixed wing aircraft
- APMRover2 - for ground vehicles and boats
- AntennaTracker - for antenna trackers

While there are a lot of common elements between different vehicle types, they are each different. For now we only have a detailed description of the code structure for the Copter code. You should go and read the [Copter Code Overview](#) wiki page now.

As you explore Copter you should also learn the following:

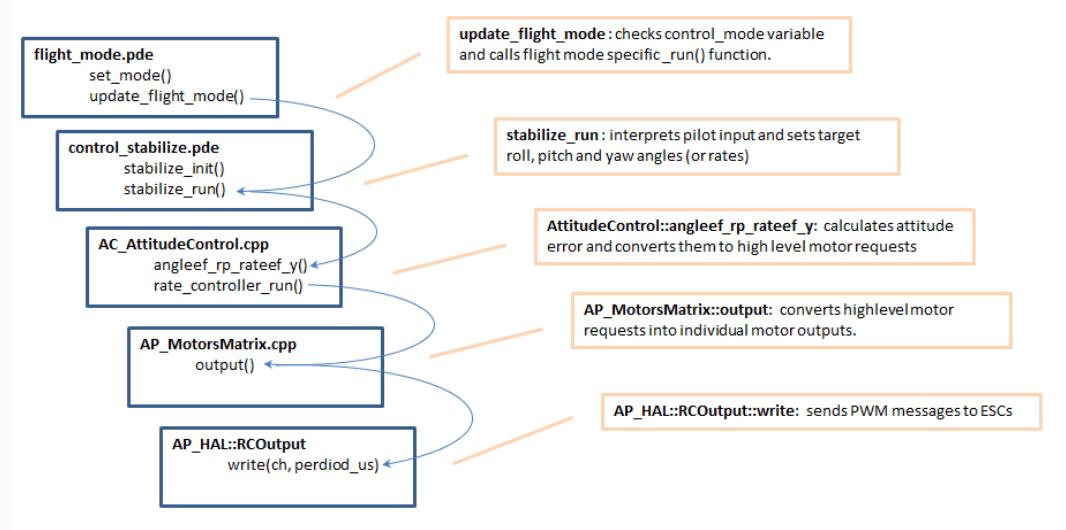
- how to [run the ArduPilot SITL system](#) to simulate a vehicle and autopilot
- how to [use gdb to debug](#) your board or SITL
- [gdb cheatsheet](#)
- how to [add parameters](#) to vehicle code or libraries

## Code Overview (Copter)

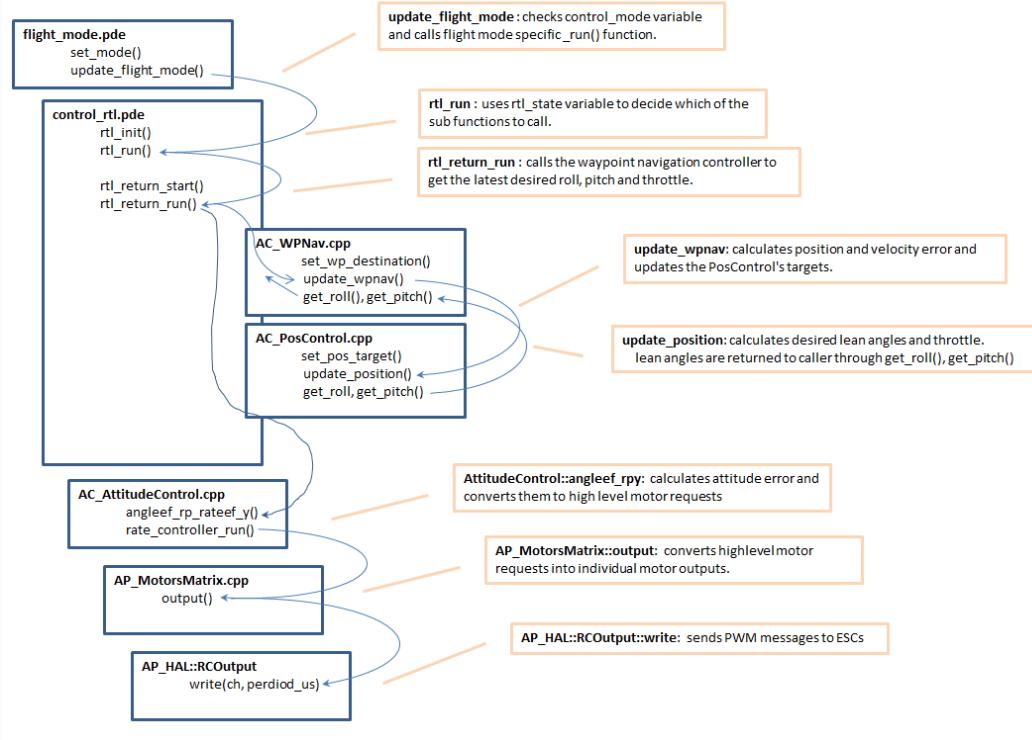
The [code](#) is made up of the main Copter code which resides in its own directory, and [the libraries](#) which are shared with Plane and Rover.

Click on the images below to see a high level view of flight-mode to motor output code:

### Manual flight modes such as Stabilize, Acro, Drift



## AutoPilot flight modes such as AltHold, RTL, Auto



Sections:

## ArduPilot Programming Libraries

The [the libraries](#) are shared with Copter, Plane and Rover. Below is a high level list of libraries and their function.

### Core libraries:

- [AP\\_AHRS](#) - attitude estimation using DCM or EKF
- [AP\\_Common](#) - core includes required by all sketches and libraries
- [AP\\_Math](#) - various math functions especially useful for vector manipulation
- [AC\\_PID](#) - PID controller library
- [AP\\_InertialNav](#) - inertial navigation library for blending accelerometer inputs with gps and baro data
- [AC\\_AttitudeControl](#) -
- [AP\\_WPNav](#) - waypoint navigation library
- [AP\\_Motors](#) - multicopter and traditional helicopter motor mixing
- [RC\\_Channel](#) - a library to more convert pwm input/output from APM\_RC into internal units such as angles
- [AP\\_HAL](#), [AP\\_HAL AVR](#), [AP\\_HAL PX4](#) - libraries to implement the “Hardware abstraction layer” which presents an identical interface to the high level code so that it can more easily be ported to different boards.

### Sensor libraries:

- [AP\\_InertialSensor](#) - reads gyro and accelerometer data, perform calibration and provides data in standard units (deg/s, m/s) to main code and other libraries
- [AP\\_RangeFinder](#) - sonar and ir distance sensor interfaced library
- [AP\\_Baro](#) - barometer interface library
- [AP\\_GPS](#) - gps interface library

- [AP\\_Compass](#) - 3-axis compass interface library
- [AP\\_OpticalFlow](#) - optical flow sensor interface library

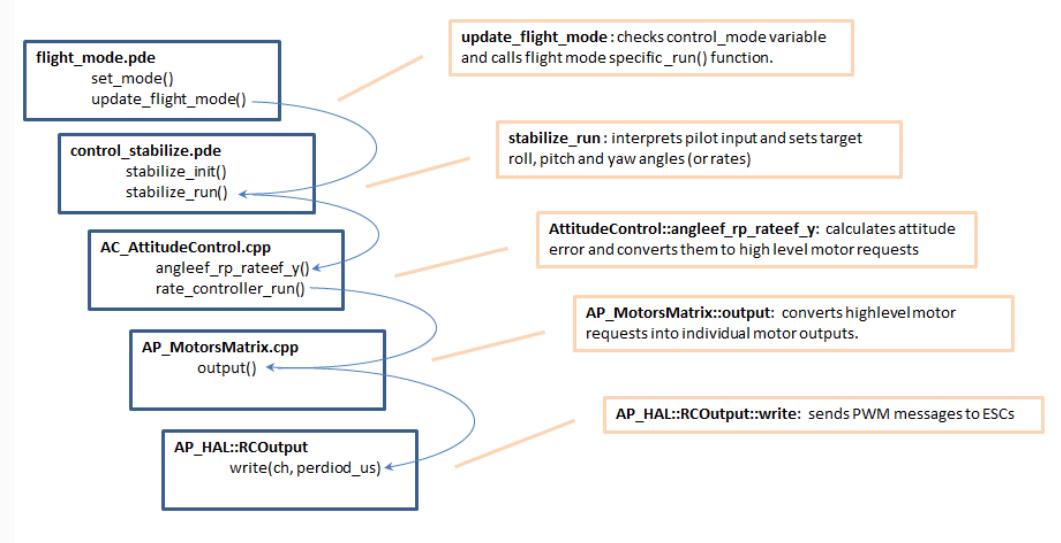
#### Other libraries:

- [AP\\_Mount](#), [AP\\_Camera](#), [AP\\_Relay](#) - camera mount control library, camera shutter control libraries
- [AP\\_Mission](#) - stores/retrieves mission commands from eeprom
- [AP\\_Buffer](#) - a simple FIFO buffer for use with inertial navigation

## Attitude Control (Copter Code Overview)

Between AC3.1.5 and AC 3.2 the attitude control logic was restructured as part of “the onion” project. The new structure is shown below.

### Manual flight modes such as Stabilize, Acro, Drift



On every update (i.e. 400hz on Pixhawk, 100hz on APM2.x) the following happens:

- the top level flight-mode.cpp’s “`update_flight_mode()`” function is called. This function checks the vehicle’s flight mode (ie. “`control_mode`” variable) and then calls the appropriate `<flight mode>_run()` function (i.e. `stabilize_run` for stabilize mode, `rtl_run` for RTL mode, etc). The `<flight mode>_run()` function can be found in the appropriately named `control_<flight mode>.cpp` file (i.e. `control_stabilize.cpp`, `control rtl.cpp`, etc).
- the `<flight mode>_run` function is responsible for converting the user’s input (found in `g.rc_1.control_in`, `g.rc_2.control_in`, etc) into a lean angle, rotation rate, climb rate, etc that is appropriate for this flight mode. For example `AltHold` converts the user’s roll and pitch input into lean angles (in degrees), the yaw input is converted into a rotation rate (in degrees per second) and the throttle input is converted to a climb rate (in cm/s).
- the last thing the `<flight mode>_run` function must do is pass these desired angles, rates etc into Attitude Control and/or Position Control libraries (these are both held in the `AC_AttitudeControl` folder).
- The `AC_AttitudeControl` library provides 5 possible ways to control the attitude of the vehicle, the most common 3 are described below.
  - `angle_ef_roll_pitch_rate_ef_yaw()` : this accepts an “earth frame” angle for roll and pitch, and an “earth frame” rate for yaw. For example providing this function `roll = -1000`, `pitch = -1500`, `yaw = 500` means lean the vehicle left to 10degrees, pitch forward to 15degrees and rotate right at 5deg/second.
  - `angle_ef_roll_pitch_yaw()` : this accepts “earth frame” angles for roll, pitch and yaw. similar to above except providing yaw of 500 means rotate the vehicle to 5 degrees east of north.
  - `rate_bf_roll_pitch_yaw()` : this accepts a “body frame” rate (in degrees/sec) for roll pitch and yaw.

For example providing this function roll = -1000, pitch = -1500, yaw = 500 would lead to the vehicle rolling left at 10deg/sec, pitching forward at 15deg/sec and rotating about the vehicle's z axis at 5 deg/sec.

After any calls to these functions are made the [AC\\_AttitudeControl::rate\\_controller\\_run\(\)](#) is called. This converts the output from the methods listed above into roll, pitch and yaw inputs which are sent to the [AP\\_Motors](#) library via it's [set\\_roll](#), [set\\_pitch](#), [set\\_yaw](#) and [set\\_throttle](#) methods. .

- The [AC\\_PosControl](#) library allows 3D position control of the vehicle. Normally only the simpler Z-axis (i.e. altitude control) methods are used because more complicated 3D position flight modes (i.e. [Loiter](#)) make use of the [AC\\_WPNav](#) library. In any case, some commonly used methods of this library include:
  - [set\\_alt\\_target\\_from\\_climb\\_rate\(\)](#) : this accepts a climb rate in cm/s and updates an absolute altitude target
  - [set\\_pos\\_target\(\)](#) : this accepts a 3D position vector which is an offset from home in cm

If any methods in AC\_PosControl are called then the flight mode code must also call the [AC\\_PosControl::update\\_z\\_controller\(\)](#) method. This will run the z-axis position control PID loops and send low-level throttle level to the [AP\\_Motors](#) library. If any xy-axis methods are called then [AC\\_PosControl::update\\_xy\\_controller\(\)](#) must be called.

- The AP\_Motors library holds the “motor mixing” code. This code is responsible for converting the roll, pitch, yaw and throttle values received from the AC\_AttitudeControl and AC\_PosControl libraries into absolute motor outputs (i.e. PWM values). So the higher level libs would make use of these functions:
  - [set\\_roll\(\)](#), [set\\_pitch\(\)](#), [set\\_yaw\(\)](#) : accepts roll, pitch and yaw values in the range of -4500 ~ 4500. These are not desired angles or even rates but rather just a value. For example set\_roll(-4500) would mean roll left as fast as possible.
  - [set\\_throttle\(\)](#) : accepts an absolute throttle value in the range of 0 ~ 1000. 0 = motors off, 1000 = full throttle.

There are different classes for each frame type (quad, Y6, traditional helicopter) but in each there is an “[output\\_armed](#)” function which is responsible for implementing the conversion of these roll, pitch, yaw and throttle values into pwm outputs. This conversion often includes implementing a “stability patch” which handles prioritising one axis of control over another when the input requests are outside the physical limits of the frame (i.e. max throttle and max roll is not possible with a quad because some motors must be less than others to cause a roll). At the bottom of the “[output\\_armed](#)” function there is a call to the [hal.rcout->write\(\)](#) which passes the desired pwm values to the AP\_HAL layer.

- The [AP\\_HAL](#) libraries (hardware abstraction layer) provides a consistent interface for all boards. In particular the [hal.rc\\_out\\_write\(\)](#) function will cause the specified PWM received from the AP\_Motors class to appear on the appropriate pwm pin out for the board.

## **Adding a New Parameter (Code Overview)**

Parameters can either be part of the main code or part of a library.

### **Adding a parameter to the main code**

**Step #1:** Find a spare slot in the Parameters class's enum in [Parameters.h](#) and add your new parameter as shown below in red. Some things to be careful of:

- Try to add the parameter close to parameters that share a similar function or worst case add to the end of the “Misc” section.
- Ensure that the section you are adding it to is not full. You can check if the section is full by counting the number of parameters in the section and ensuring that the last element is less than the next

sections first element. I.e. the Misc section's first parameter is #20. my\_new\_parameter is #36. If the next section began at #36 we would not be able to add the new parameter here.

- Do not add a parameter in the middle of a group thus causing the slots for existing parameters to change
- Do not use slots with “deprecated” or “remove” comments beside them because some users may still have the defaults for these old parameters in their eeprom so your new parameter’s default value could be set strangely.

```
enum {
    // Misc
    //
    k_param_log_bitmask = 20,           // *** Deprecated - remove
    k_param_log_last_filenumber,        // with next eeprom number
                                         // change
    k_param_toy_yaw_rate,              // THOR The memory
                                         // location for the
                                         // Yaw Rate 1 = fast,
                                         // 2 = med, 3 = slow

    k_param_crosstrack_min_distance,    // deprecated - remove with next eeprom number change
    k_param_rssi_pin,
    k_param_throttle_accel_enabled,     // deprecated - remove
    k_param_wp_yaw_behavior,
    k_param_acro_trainer,
    k_param_pilot_velocity_z_max,
    k_param_circle_rate,
    k_param_sonar_gain,
    k_param_ch8_option,
    k_param_arming_check_enabled,
    k_param_sprayer,
    k_param_angle_max,
    k_param_gps_hdop_good,            // 35
    k_param_my_new_parameter,          // 36
```

**Step #2:** Declare the variable within the Parameters class somewhere below the enum. Possible types include AP\_Int8, AP\_Int16, AP\_Float, AP\_Int32 and AP\_Vector3 (Note: unsigned integers are not currently supported). The name of the variable should be the same as the new enum but with the “k\_param\_” prefix removed.

```
// 254,255: reserved
};

AP_Int16      format_version;
AP_Int8       software_type;

// Telemetry control
//
AP_Int16      sysid_this_mav;
AP_Int16      sysid_my_gcs;
AP_Int8       serial3_baud;
AP_Int8       telem_delay;

AP_Int16      rtl_altitude;
AP_Int8       sonar_enabled;
AP_Int8       sonar_type;      // 0 = XL, 1 = LV,
                           // 2 = XLL (XL with 10m range)
                           // 3 = HRLV
AP_Float      sonar_gain;
AP_Int8       battery_monitoring; // 0=disabled, 3=voltage only,
                                // 4=voltage and current
AP_Float      volt_div_ratio;
AP_Float      curr_amp_per_volt;
AP_Int16      pack_capacity;    // Battery pack capacity less reserve
AP_Int8       failsafe_battery_enabled; // battery failsafe enabled
AP_Int8       failsafe_gps_enabled; // gps failsafe enabled
AP_Int8       failsafe_gcs;      // ground station failsafe behavior
AP_Int16      gps_hdop_good;   // GPS Hdop value below which represent a good position
AP_Int16      my_new_parameter; // my new parameter's description goes here
```

**Step #3:** Add the variable declaration to the var\_info table in [Parameters.cpp](#)

```
// @Param: MY_NEW_PARAMETER
// @DisplayName: My New Parameter
// @Description: A description of my new parameter goes here
// @Range: -32768 32767
// @User: Advanced
GSCALAR(my_new_parameter, "MY_NEW_PARAMETER", MY_PARAMETER_DEFAULT),
```

The @Param ~ @User comments are used by the ground station (i.e. Mission Planner) to display advice to the user and limit the values that the user may set the parameter to.

The parameter name (i.e. "MY\_NEW\_PARAMETER") is limited to 16 characters.

**Step #4:** Add the parameters default to [config.h](#)

```
#ifndef MY_NEW_PARAMETER_DEFAULT
#define MY_NEW_PARAMETER_DEFAULT      100      // default value for my new parameter
#endif
```

You're done! The new parameter can be access from the main code (not the libraries) as g.my\_new\_parameter.

**Adding a parameter to a library**

Parameters can also be added to libraries by following these steps which use the [AP\\_Compass](#) library as an example.

**Step #1:** Add the new class variable to the top level .h file (i.e. [Compass.h](#)). Possible types include AP\_Int8, AP\_Int16, AP\_Float, AP\_Int32 and AP\_Vector3f. Also add the default value you'd like for the parameter (we will use this in step #2)

```
#define MY_NEW_PARAM_DEFAULT      100

class Compass
{
public:
    int16_t product_id;           /// product id
    int16_t mag_x;               ///< magnetic field strength along the X axis
    int16_t mag_y;               ///< magnetic field strength along the Y axis
    int16_t mag_z;               ///< magnetic field strength along the Z axis
    uint32_t last_update;        ///< micros() time of last update
    bool healthy;                ///< true if last read OK

    /// Constructor
    ///
    Compass();

protected:
    AP_Int8 _orientation;
    AP_Vector3f _offset;
    AP_Float _declination;
    AP_Int8 _use_for_yaw;         ///<enable use for yaw calculation
    AP_Int8 _auto_declination;   ///<enable automatic declination code
    AP_Int16 _my_new_lib_parameter; // description of my new parameter
};
```

**Step #2:** Add the variable to the var\_info table in the .cpp file (i.e. [Compass.cpp](#)) including @Param ~ @Increment comments to allow the GCS to display the description to the user and to control the min and max values set from the ground station. When adding the new parameter be careful that:

- The slot (i.e. “9” below) is one higher than the previous variable.
- the parameter’s name (i.e. MY\_NEW\_P) length is less than 16 including the object’s prefix that will be added. I.e. the compass object’s prefix is “COMPASS\_”.

```
const AP_Param::GroupInfo Compass::var_info[] = {
    // index 0 was used for the old orientation matrix

    // @Param: OFS_X
    // @DisplayName: Compass offsets on the X axis
    // @Description: Offset to be added to the compass x-axis values to compensate for metal in the frame
    // @Range: -400 400
    // @Increment: 1

    <snip>

    // @Param: ORIENT
    // @DisplayName: Compass orientation
    // @Description: The orientation of the compass relative to the autopilot board.
    // @Values: 0:None,1:Yaw45,2:Yaw90,3:Yaw135,4:Yaw180,5:Yaw225,6:Yaw270,7:Yaw315,8:Roll180
    AP_GROUPINFO("ORIENT", 8, Compass, _orientation, ROTATION_NONE),

    // @Param: MY_NEW_P
    // @DisplayName: My New Library Parameter
    // @Description: The new library parameter description goes here
    // @Range: -32768 32767
    // @User: Advanced
    AP_GROUPINFO("MY_NEW_P", 9, Compass, _my_new_lib_parameter, MY_NEW_PARAM_DEFAULT),

    AP_GROUPEnd
};
```

The parameter can be accessed from within the library as \_my\_new\_lib\_parameter. Note that we made the parameter “protected” so it cannot be access from outside the class. If we’d made it public then it would have been accessible to the main code as well as “compass.\_my\_new\_lib\_parameter”.

**Step #3:** Add a declaration for var\_info to the public section of the .h file of new library class in addition to the definition in the .cpp file:

```
static const struct AP_Param::GroupInfo var_info[];
```

**Step #4:** If the class is a completely new addition to the code (as opposed to an existing class like AP\_Compass), it should be added to the main vehicle’s var\_info table in the [Parameters.cpp](#) file (it’s order in the var\_info table is not important). Below in red where the Compass class appears.

```
const AP_Param::Info var_info[] = {
    // @Param: SYSID_SW_MREV
    // @DisplayName: Eeprom format version number
    // @Description: This value is incremented when changes are made to the eeprom format
    // @User: Advanced
    GSCALAR(format_version, "SYSID_SW_MREV", 0),
    <snip>

    // @Group: COMPASS_
    // @Path: ../libraries/AP_Compass/Compass.cpp
    GOBJECT(compass, "COMPASS_", Compass),

    <snip>
    // @Group: INS_
    // @Path: ../libraries/AP_InertialSensor/AP_InertialSensor.cpp
    GOBJECT(ins, "INS_", AP_InertialSensor),

    AP_VAREND
};
```

**Step #5:** If the class is a completely new addition to the code, also add k\_param\_my\_new\_lib to the

enum in *Parameters.h* <<https://github.com/ArduPilot/ardupilot/blob/master/ArduCopter/Parameters.h>>, where `my_new_lib` is the first argument to the `GOBJECT` declaration in *Parameters.cpp*. Read the comments above the enum to understand where to place the new value, as order is important here.

## Adding a New Flight Mode (Copter Code Overview)

This section covers the basics of how to create a new high level flight mode (i.e. equivalent of Stabilize, Loiter, etc) which is the top level of “the onion” as described on the [Attitude](#) page. This page unfortunately can’t provide all the information on what you may need to do to create your ideal flight mode but hopefully it’s a start.

1. Create the `#define` for the new flight mode in [defines.h](#). and increase the `NUM_MODES` by 1.

```
// Auto Pilot modes
// -----
#define STABILIZE 0 // hold Level position
#define ACRO 1 // rate control
#define ALT_HOLD 2 // AUTO control
#define AUTO 3 // AUTO control
#define GUIDED 4 // AUTO control
#define LOITER 5 // Hold a single location
#define RTL 6 // AUTO control
#define CIRCLE 7 // AUTO control
#define LAND 9 // AUTO control
#define OF_LOITER 10 // Hold a single location using optical flow sensor
#define DRIFT 11 // DRIFT mode (Note: 12 is no longer used)
#define SPORT 13 // earth frame rate control
#define FLIP 14 // flip the vehicle on the roll axis
#define AUTOTUNE 15 // autotune the vehicle's roll and pitch gains
#define POSHOLD 16 // position hold with manual override
#define NEWFLIGHTMODE 17 // new flight mode description
#define NUM_MODES 18
```

2. Create a new `control_<new flight mode>` sketch based on a similar flight mode such as [control\\_stabilize.cpp](#) or [control\\_loiter.cpp](#). This new file should have an `_init()` function and `_run()` function.

```
/// -*- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: nil -*-
/*
 * control_newflightmode.cpp - init and run calls for new flight mode
 */

// newflightmode_init - initialise flight mode
static bool newflightmode_init(bool ignore_checks)
{
    // do any required initialisation of the flight mode here
    // this code will be called whenever the operator switches into this mode

    // return true if initialisation is successful, false if it fails
    // if false is returned here the vehicle will remain in the previous flight mode
    return true;
}

// newflightmode_run - runs the main controller
// will be called at 100hz or more
static void newflightmode_run()
{
    // if not armed or throttle at zero, set throttle to zero and exit immediately
    if(!motors.armed() || g.rc_3.control_in <= 0) {
        attitude_control.relax_bf_rate_controller();
        attitude_control.set_yaw_target_to_current_heading();
        attitude_control.set_throttle_out(0, false);
        return;
    }

    // convert pilot input into desired vehicle angles or rotation rates
```

```

// g.rc_1.control_in : pilots roll input in the range -4500 ~ 4500
// g.rc_2.control_in : pilot pitch input in the range -4500 ~ 4500
// g.rc_3.control_in : pilot's throttle input in the range 0 ~ 1000
// g.rc_4.control_in : pilot's yaw input in the range -4500 ~ 4500

// call one of attitude controller's attitude control functions like
// attitude_control.angle_ef_roll_pitch_rate_yaw(roll angle, pitch angle, yaw rate);

// call position controller's z-axis controller or simply pass through throttle
// attitude_control.set_throttle_out(desired throttle, true);
}

```

3. Add declarations in [Copter.h](#) for the new `_init()` function and `_run()` functions:

```

bool newflightmode_init(bool ignore_checks);
void newflightmode_run();

```

4. Add a case for the new mode to the `set_mode()` function in [flight\\_mode.cpp](#) to call the `above _init()` function.

```

// set_mode - change flight mode and perform any necessary initialisation
static bool set_mode(uint8_t mode)
{
    // boolean to record if flight mode could be set
    bool success = false;
    bool ignore_checks = !motors.armed(); // allow switching to any mode if disarmed. We rely on the
                                         // arming check to perform

    // return immediately if we are already in the desired mode
    if (mode == control_mode) {
        return true;
    }

    switch(mode) {
        case ACRO:
            #if FRAME_CONFIG == HELI_FRAME
                success = heli_acro_init(ignore_checks);
            #else
                success = acro_init(ignore_checks);
            #endif
            break;

        case NEWFLIGHTMODE:
            success = newflightmode_init(ignore_checks);
            break;
    }
}

```

5. Add a case for the new mode to the `update_flight_mode()` function in [flight\\_mode.cpp](#) to call the above `_run()` function.

```

// update_flight_mode - calls the appropriate attitude controllers based on flight mode
// called at 100hz or more
static void update_flight_mode()
{
    switch (control_mode) {
        case ACRO:
            #if FRAME_CONFIG == HELI_FRAME
                heli_acro_run();
            #else
                acro_run();
            #endif
            break;
        case NEWFLIGHTMODE:
            success = newflightmode_run();
            break;
    }
}

```

6. Add the string to print out the flight mode to the `print_flight_mode()` function in `flight_mode.cpp`.

```
static void
print_flight_mode(AP_HAL::BetterStream *port, uint8_t mode)
{
    switch (mode) {
    case STABILIZE:
        port->print_P(PSTR("STABILIZE"));
        break;
    case NEWFLIGHTMODE:
        port->print_P(PSTR("NEWFLIGHTMODE"));
        break;
    }
```

7. Add the new flight mode to the list of valid `@Values` for the `FLTMODE1 ~ FLTMODE6` parameters in `Parameters.cpp`.

```
// @Param: FLTMODE1
// @DisplayName: Flight Mode 1
// @Description: Flight mode when Channel 5 pwm is 1230, <= 1360
// @Values:
0:Stabilize,1:Acro,2:AltHold,3:Auto,4:Guided,5:Loiter,6:RTL,7:Circle,8:Position,9:Land,10:OF_Loiter,11:To

// @User: Standard
GSCALAR(flight_mode1, "FLTMODE1", FLIGHT_MODE_1),

// @Param: FLTMODE2
// @DisplayName: Flight Mode 2
// @Description: Flight mode when Channel 5 pwm is >1230, <= 1360
// @Values:
0:Stabilize,1:Acro,2:AltHold,3:Auto,4:Guided,5:Loiter,6:RTL,7:Circle,8:Position,9:Land,10:OF_Loiter,11:To

// @User: Standard
GSCALAR(flight_mode2, "FLTMODE2", FLIGHT_MODE_2),
```

8. Raise a request in the [Mission Planner's Issue List](#) if you wish the new flight mode to appear in the Mission Planner's HUD and Flight Mode set-up.



## Scheduling Code to Run Intermittently (Code Overview)

This page describes how you can schedule some new piece of code you have written to run intermittently.

## Running your code with the scheduler

The most flexible way to run your code at a given interval is to use the scheduler. This can be done by adding your new function to the `scheduler_tasks` array in [ArduCopter.cpp](#). Note that there are actually two task lists, the [upper list](#) is for high speed CPUs (i.e. Pixhawk) and the [lower list](#) is for slow CPUs (i.e. APM2).

Adding a task is fairly simple, just create a new row in the list (higher in the list means high priority). The first column holds the function name, the 2nd is a number of 2.5ms units (or 10ms units in case of APM2). So if you wanted the function executed at 400hz this column would contain "1", if you wanted 50hz it would contain "8". The final column holds the number of microseconds (i.e. millions of a second) the function is expected to take. This helps the scheduler guess whether or not there is enough time to run your function before the main loop starts the next iteration.

```
/*
scheduler table - all regular tasks apart from the fast_loop()
should be listed here, along with how often they should be called
(in 10ms units) and the maximum time they are expected to take (in
microseconds)
*/
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
    { update_GPS,          2,     900 },
    { update_nav_mode,     1,     400 },
    { medium_loop,         2,     700 },
    { update_altitude,     10,    1000 },
    { fifty_hz_loop,       2,     950 },
    { run_nav_updates,     10,    800 },
    { slow_loop,           10,    500 },
    { gcs_check_input,     2,     700 },
    { gcs_send_heartbeat,  100,   700 },
    { gcs_data_stream_send, 2,    1500 },
    { gcs_send_deferred,   2,    1200 },
    { compass_accumulate,  2,     700 },
    { barometer_accumulate, 2,     900 },
    { super_slow_loop,     100,   1100 },
    { my_new_function,     10,    200 },
    { perf_update,         1000,  500 }
};
```

## Running your code as part of one of the loops

Instead of creating a new entry in the scheduler task list, you can add your function to one of the existing timed loops. There is no real advantage to this approach over the above approach except in the case of the fast-loop. Adding your function to the fast-loop will mean that it runs at the highest possible priority (i.e. it is nearly 100% guaranteed to run at 400hz).

- `fast_loop` : runs at 100hz on APM2, 400hz on Pixhawk
- `fifty_hz_loop` : runs at 50hz
- `ten_hz_logging_loop`: runs at 10hz
- `three_hz_loop`: runs at 3.3hz
- `>one_hz_loop` : runs at 1hz

So for example if you want your new code to run at 10hz you could add it to one of the case statements in the `ten_hz_logging_loop()` function found in [ArduCopter.cpp](#).

```
// ten_hz_logging_loop
// should be run at 10hz
static void ten_hz_logging_loop()
{
```

```

if (g.log_bitmask & MASK_LOG_ATTITUDE_MED) {
    Log_Write_Attitude();
}
if (g.log_bitmask & MASK_LOG_RCIN) {
    DataFlash.Log_Write_RCIN();
}
if (g.log_bitmask & MASK_LOG_RCOUT) {
    DataFlash.Log_Write_RCOUT();
}
if ((g.log_bitmask & MASK_LOG_NTUN) && mode_requires_GPS(control_mode)) {
    Log_Write_Nav_Tuning();
}

// your new function call here
my_new_function();
}

```

## Adding a new MAVLink Message (Code Overview)

Data and commands are passed between the ground station (i.e mission planner, Droid Planner, etc) using the [MAVLink protocol](#) over a serial interface. This page provides some high level advice for adding a new MAVLink message.

These instructions have only been tested on Linux (to be precise a VM running Ubuntu on a Windows machine). Instructions for setting up a VM are on the [SITL page](#). If you can run SITL, you should be able to follow the advice here. These instructions will not run natively on Windows or a Mac.

**Step #1:** Ensure you have the latest ArduPilot code installed. Also check mavproxy. Mavproxy can be updated by running this command in a Terminal window:

```
sudo pip install --upgrade mavproxy
```

**Step #2:** Decide what type of message you want to add and how it will fit in with the existing [MAVLink messages](#).

For example you might want to send a new navigation command to the vehicle so that it can perform a trick (like a flip) in the middle of a mission (i.e. in AUTO mode). In this case you would need a new MAV\_CMD\_NAV\_TRICK similar to the MAV\_CMD\_NAV\_WAYPOINT definition (search for "MAV\_CMD\_NAV\_WAYPOINT" in the [MAVLink messages](#) page).

Alternatively you may want to send down a new type of sensor data from the vehicle to the ground station. Perhaps similar to the [SCALED\\_PRESSURE](#) message.

**Step #3:** Add the new message definition to the [common.xml](#) or [ardupilotmega.xml](#) file in the mavlink submodule.

If this command will hopefully be added to the MAVLink protocol then it should be added to the `./modules/mavlink/message_definitions/v1.0/common.xml` file. If it is only for your personal use or only for use with Copter, Plane, Rover then it should be added to the `ardupilotmega.xml` file.

**Step #4:** Starting in Jan 2016 the source code is automatically generated when you compile the project but before that date you would cd to the ardupilot directory and then run this command to manually generate it.

```
./libraries/GCS_MAVLink/generate.sh
```

**Step #5:** Add functions to the main vehicle code to handle sending or receiving the command to/from the ground station. A compile will be needed (ie. make px4-v2) to generate the mavlink packet code so make

sure to do that after editing the xml file. The mavlink generation happens first so it doesn't matter if the project compilation is successful or not due to other source code changes.

The top level of this code will most likely be in the vehicle's [GCS\\_MAVLink.cpp](#) file or in the [..libraries/GCS\\_MAVLink/GCS](#) class.

In the first example where we want to add support for a new navigation command (i.e. a trick) the following would be required:

- Extend the [AP\\_Mission](#) library's `mission_cmd_to_mavlink()` and `mavlink_to_mission_cmd()` functions to convert the MAVProxy command into an `AP_Mission::Mission_Command` structure.
- Add a new case to the vehicle's [commands\\_logic.cpp](#)'s `start_command()` and `verify_command()` functions to check for the arrival of the new `MAV_CMD_NAV_TRICK`. These should call two new functions that you create called `do_trick()` and `verify_trick()` (see below).
- Create these two new functions, `do_trick()` and `verify_trick()`, that somehow command the vehicle *to perform the trick* (perhaps by calling another function in [control\\_auto.cpp](#) that sets the `auto_mode` variable and then calls a new `auto_trick_start()` function). The `do_trick()` function will be called when the command is first invoked. The `verify_trick()` will be called at 10hz (or higher) repeatedly until the trick is complete. The `verify_trick()` function should return true when the trick has been completed.

**Step #6:** Consider contributing your code back to the main code base. Email the [drones-discuss email list](#) and/or [raise a pull request](#). If you raise a pull request it is best to separate the change into at least two separate commits. One commit for the changes to the .xml files (i.e Step #3) and another for the changes to the vehicle code.

## Adding Support for a new MAVLink Gimbal

This page covers how ArduPilot interacts with a MAVLink enabled gimbal. It is meant more as a guide on how gimbal manufacturers can make their gimbals work with ArduPilot with minimum code changes to ArduPilot .

The [SToRM32 gimbal](#) is a good reference as it supports MAVLink.

### Messages the gimbal should support

1. The gimbal should listen on the serial port for a [HEARTBEAT](#) message from the vehicle. It can generally assume that the first heart beat it receives will be from the vehicle but to be certain you can check the "type" field to be sure it's something sensible (i.e. `MAV_TYPE = 1` for fixed wing, 2 for quadcopters but 6 for GCSs should be ignored).

This heart beat will contain the vehicle's system-id and component-id. The gimbal should adopt the same system-id but should use a component-id of `MAV_COMP_ID_GIMBAL` (i.e. 154) for all its future messages. In fact, any unique component-id can be used as long as it's not zero nor the component id of the vehicle or any other device on the vehicle.

2. The gimbal should send a [HEARTBEAT](#) message out the serial port at approximately 1hz
  - the system-id and component-id should be as mentioned above.
  - "type" should be `MAV_TYPE_GIMBAL` (i.e. 26).
  - "autopilot" is not used so can be set to anything including `MAV_AUTOPILOT_GENERIC` (i.e. 0)
  - "base\_mode" and "custom\_mode" are not used so can be set to anything (perhaps 0 is best)
  - "system\_status" should be set to "MAV\_STATE\_ACTIVE" once the gimbal is ready to accept attitude targets
3. To support reading/writing parameter values from the ground station the gimbal should implement these message:
  - [PARAM\\_REQUEST\\_READ](#) - if the gimbal receives this message and "target\_system" and

- “target\_component” values match the gimbal’s system-id and component-id, it should respond with a [PARAM\\_VALUE](#) message which contains the value of the parameter specified by the “param\_id” field (simply an enum, the gimbal can assign whatever enum it wishes to each of its internal parameters)
- [PARAM\\_REQUEST\\_LIST](#) - respond to this message by sending a [PARAM\\_VALUE](#) message for every parameter within the gimbal.
  - [PARAM\\_SET](#) - respond to this by setting the internal variable to the value in the “param\_value” field.
4. ArduPilot will send angle requests to the gimbal via MAVLink which will arrive as [COMMAND\\_LONG](#) messages with the “command” field set to MAV\_CMD\_DO\_MOUNT\_CONTROL (i.e 205).
    - Param #1 contains desired pitch in degrees
    - Param #2 contains desired roll in degrees
    - Param #3 contains desired yaw in degrees but note that the yaw is in relation to the front of the vehicle so “0” is straight ahead, “90” is to the right, “-90” is to the left.
  5. If the gimbal needs extra data from the vehicle it can request it using the [REQUEST\\_DATA\\_STREAM](#) message.
    - the target system and component id should be for the vehicle.
    - “req\_stream\_id” can be any values in the MAV\_DATA\_STREAM enum. Some useful values are:
      - MAV\_DATA\_STREAM\_POSITION will cause the vehicle to send [GLOBAL\\_POSITION\\_INT](#) messages which includes lat, lon, alt, velocity (3d) and heading
      - MAV\_DATA\_STREAM\_EXTRA1 will send the [MSG\\_ATTITUDE](#) which includes euler angles for roll, pitch, yaw

## Testing

To test the system, you should be able to connect the gimbal to a Pixhawk as if it’s a SToRM32 gimbal. In particular check the “[Set-up through the Mission Planner \(MALVink protocol\)](#)” section.

## Building the code

### Note

Code building has changed for newer releases to use [waf](#) build tools, replacing make.

In most cases the build dependencies described for [make](#) are the same, the only part of the instructions changes is the issue of the [waf](#) build command.

see <https://github.com/ArduPilot/ardupilot/blob/master/BUILD.md>

The linked articles below explain how to build ArduPilot for different target hardware on the supported development environments (Linux, Windows, Mac OSX). The included links also cover building the code for ground stations.

## Plane, Copter, Rover

### Windows users:

- [Building ArduPilot on Windows10 with Bash on Ubuntu on Windows](#)
- [Building ArduPilot with Arduino for Windows](#)
- [Pixhawk/PX4 on Windows with Make](#)
- [Editing & Building with Atmel Studio or Visual Studio](#)

### MacOS users:

- [APM2.x on MacOS with Arduino](#)
- [Pixhawk/PX4 on MacOs with Make](#)

**Linux users:**

- [APM2.x on Linux with Make](#)
- [Pixhawk/PX4 on Linux with Make](#)
- [Beaglebone Black with Make](#)
- [Building for Flymaple on Linux](#)
- [Building for NAVIO+ on RPi2](#)
- [Building for NAVIO2 on RPi3](#)
- [Building for Erle-Brain](#)
- [Building for Erle-Brain 2](#)
- [Building for Bebop on Linux](#)
- [Building for Bebop 2 on Linux](#)

**IDE/Cross platform**

- [Building With Make \(Win, Mac, Linux\)](#)

**Related information**

- [Git Submodules](#)

**Mission Planner**

- [Building Mission Planner with Visual Studio](#)

**Building ArduPilot for Pixhawk/PX4 on Linux with Make**

This article shows how to build ArduPilot for Pixhawk 2, Pixhawk and PX4 on Linux with *Make*.

**Note**

The commands for building Pixhawk 2 and Pixhawk are identical (`make px4-v2`). Building for PX4 is the same except that `make px4-v1` is used.

**Quick start**

For Ubuntu, follow these steps to build the code. For other distributions, see the advanced instructions below.

**Setup¶**

Install git:

```
sudo apt-get -qq -y install git
```

Clone the source:

```
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

Run the `install-prereqs-ubuntu.sh` script:

```
Tools/scripts/install-prereqs-ubuntu.sh -y
```

Reload the path (log-out and log-in to make permanent):

```
. ~/.profile
```

## Build¶

The commands below show how to build for Pixhawk2/Pixhawk on the different platforms. To build for PX4 replace `make px4-v2` with `make px4-v1`.

Build for Copter:

```
cd ArduCopter
make px4-v2
```

Build for Plane:

```
cd ArduPlane
make px4-v2
```

Build for Rover:

```
cd APMrover2
make px4-v2
```

Build for Antenna Tracker:

```
cd AntennaTracker
make px4-v2
```

## Advanced

To build for a Pixhawk2/Pixhawk/PX4 target on Linux you need the following tools and git repositories:

- The gcc-arm cross-compiler from [here](#)
- The ardupilot git repository from [github.com/ArduPilot/ardupilot](https://github.com/ArduPilot/ardupilot)
- gnu make, gawk and associated standard Linux build tools
- On Ubuntu you will need to install the genromfs package.
- On a 64 bit system you will also need to have installed libc6-i386.

## Permissions¶

You need to make your user a member of the dialout group:

```
sudo usermod -a -G dialout $USER
```

You will need to log out and then log back in for the group change to take effect.

Also, it's worth mentioning here that you want to ensure that the modemmanager package is not installed and the modem-manager process is not running.

## Directory Layout¶

The ardupilot, PX4NuttX and PX4Firmware git checkouts all need to be in the same directory. The makefile looks in the directory above the ardupilot directory to find the PX4NuttX and PX4Firmware trees.

## Compiler¶

You need the specific gcc-arm cross-compiler linked above. You need to unpack it:

```
tar -xjvf gcc-arm-none-eabi-4_6-2012q2-20120614.tar.bz2
```

and then add the bin directory from the tarball to your \$PATH by editing the \$HOME/.bashrc file and adding a line like this to the end:

```
export PATH=$PATH:/home/your_username/bin/gcc-arm-none-eabi-4_6-2012q2/bin
```

#### ccache for faster builds¶

Installing ccache will speed up your builds a lot. Once you install it (for example with “sudo apt-get install ccache”) you should link the compiler into /usr/lib/ccache like this:

```
cd /usr/lib/ccache
sudo ln -s /usr/bin/ccache arm-none-eabi-g++
sudo ln -s /usr/bin/ccache arm-none-eabi-gcc
```

Then add /usr/lib/ccache to the front of your \$PATH

#### Building¶

Once you have the 3 git trees and compiler setup you do the build in your vehicle directory. For example, if building Plane then do this:

```
cd ArduPlane
make px4
```

That will build two files **ArduPlane-v1.px4** and **ArduPlane-v2.px4**. The v1 file is for PX4v1, the v2 file is for PX4v2 (the Pixhawk).

You can also build for just one board by using “make px4-v1” or “make px4-v2”.

The first time you build it will take quite a long time as it builds the px4 archives. Subsequent builds will be faster (especially if you setup ccache correctly).

#### Loading firmware¶

To load the firmware onto the board use

```
make px4-v1-upload
```

or

```
make px4-v2-upload
```

After it says “waiting for bootloader” plugin your PX4 on USB.

If upload consistently fails in the erase step then check if you are running ‘modemmanager’ which can take control of the PX4 USB port. Removing modemmanager can help.

#### Cleaning¶

If there have been updates to the PX4NuttX or PX4Firmware git submodules you may need to do a full clean build. To do that use:

```
make px4-clean
```

that will remove the *PX4NuttX* archives so you can do a build from scratch

## Building ArduPilot for Pixhawk/PX4 on Windows with Make

This article shows how to build ArduPilot for Pixhawk 2, Pixhawk and PX4 on Windows with *Make*.

### Note

The commands for building Pixhawk 2 and Pixhawk are identical (`make px4-v2`). Building for PX4 is the same except that `make px4-v1` is used.

### Build instructions

1. Install [GitHub for Windows](#)

2. Ensure your github settings are set to leave line endings untouched.

- The “Git Shell (or Bash)” terminal was also installed when you installed Git. Click on your new “Git Shell (or Bash)” Icon and type in the following in the Git “MINGW32” Terminal window:

```
git config --global core.autocrlf false
```

3. Clone the ardupilot repository onto your machine:

- Go to the [GitHub/ArduPilot/ardupilot](#) web page and click the **Clone in Desktop** button
- Warning: be careful that the directory path is less than about 50 characters. For example “C:\Users\rmackay9\Documents\GitHub\ardupilot” is short enough but “C:\Users\rmackay9\Documents\GitHub\rmackay9-ardupilot” is too long. This limit is because during compiling temporary files are created with much much longer paths which can exceed Windows’ 260 character path limit.

### Initialise and update submodules

```
git submodule update --init --recursive
```

Download and install the *PX4 toolchain* by running the [pixhawk\\_toolchain\\_installer\\_latest.exe](#)

Open the *PX4Console* and navigate to the target vehicle directory:

- Start the *PX4Console*. This can be found under **Start | All Programs | PX4 Toolchain** (Windows 7 machine) or you can directly run **C:\px4\toolchain\msys\1.0\px4\_console.bat**
- Navigate to the vehicle-specific ArduPilot directory in the *PX4Console*. For example, to build Copter, navigate to:

```
cd /c/Users<username>/Documents/GitHub/ardupilot/ArduCopter
```

Build the firmware by entering one of the following commands:

<code>make px4-v2</code>	Build the Pixhawk2/Pixhawk firmware (identical) for a quad
<code>make px4-v2-hexa</code>	Build the Pixhawk firmware for a hexacopter. # Other supported suffixes include “octa”, “octa-quad”, “tri”

	and "heli".
	# More can be found in "mk/targets.mk" under FRAMES
<code>make px4</code>	Build both PX4 and PixHawk firmware for a quadcopter
<code>make clean</code>	"clean" the ardupilot directory
<code>make px4-clean</code>	"clean" the PX4Firmware and PX4NuttX directories so the next build will completely rebuild them
<code>make px4-v2-upload</code>	Build and upload the Pixhawk firmware for a quad (i.e. no need to do step #7 below)

The firmware will be created in the **ArduCopter** directory with the **.px4** file extension.

```
MINGW32:/c/px4/ardupilot/ArduCopter
make[2]: Leaving directory '/c/px4/PX4Firmware/makefiles/build/c/px4/PX4Firmware'
/src/systemcmds/top'
CMDS:  /c/px4/PX4Firmware/makefiles//build/builtin_commands.c
CC:    /c/px4/PX4Firmware/makefiles//build/builtin_commands.c
LINK:  /c/px4/PX4Firmware/makefiles//build/firmware.elf
BIN:   /c/px4/PX4Firmware/makefiles//build/firmware.bin
%% Generating /c/px4/PX4Firmware/makefiles//build/firmware.px4
make[1]: Leaving directory '/tmp/ArduCopter_build'
PX4 ArduCopter Firmware is in ArduCopter.px4
Gary@Gary-Dell-PC /c/px4/ardupilot/ArduCopter
$ -
```

- Upload the firmware using the **Mission Planner Initial Setup | Install Firmware** screen's **Load custom firmware** link

#### Note

ArduPilot imports addition projects ([PX4Firmware](#), [PX4NuttX](#), [uavcan](#)) as *git submodules* when you build the project. If you built the project before the change to submodules you may get errors. See [Git Submodules](#) for troubleshooting information.

#### Note

You can ignore any messages regarding PX4Firmware and PX4NuttX hashes. Those are useful labels for developers but optional and sometimes the build system can't find them on your system. As long as it says "Firmware is in..." followed by a .px4 file then you have a successful build which you can safely load onto your aircraft.

#### Hints for speeding up compile time

Anti virus protection is likely to slow the compile times especially for PX4 so it is recommended that the folders containing the ArduPilot source code is excluded from your virus protections real-time scan.

The first scan after a `make px4-clean` will be very slow as it rebuilds everything

## Building ArduPilot for Pixhawk/PX4 on Mac with Make

This article shows how to build ArduPilot for Pixhawk 2, Pixhawk and PX4 on Mac OS X (ver 10.6 onwards) with *Make*.

#### Note

The commands for building Pixhawk 2 and Pixhawk are identical (`make px4-v2`). To build for PX4 replace `make px4-v2` with `make px4-v1` in the instructions below. #. Install [Homebrew](#)for Mac OS X

1. Install xcode and say YES to install Command Line Tools

```
xcode-select --install
```

2. Install the following packages using brew

```
brew tap PX4/homebrew-px4
brew update
brew install genromfs
brew install gcc-arm-none-eabi
```

3. Install the latest version of awk using brew (make sure **/usr/local/bin** takes precedence in your path):

```
brew install gawk
```

4. Install *pip* and *pyserial* using the following commands:

```
sudo easy_install pip
sudo pip install pyserial
```

5. Now create your directory and install all the software:

```
mkdir -p px4
cd px4
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

#### Note

##### **PX4Firmware**,

**PX4NuttX** and **uavcan** are automatically imported as **Git Submodules** when you build a vehicle.

6. Build the firmware by entering one of the following commands:

<code>make px4-v2</code>	Build the Pixhawk2/Pixhawk firmware (identical) for a quad
<code>make px4-v2-hexa</code>	Build the Pixhawk firmware for a hexacopter. # Other supported suffixes include "octa", "tri" and "heli". # More can be found in "mk/tags.mk" under FRAMES
<code>make px4</code>	Build both PX4 and PixHawk firmware for a quadcopter
<code>make clean</code>	"clean" the ardupilot directory
<code>make px4-clean</code>	"clean" the PX4Firmware and PX4NuttX directories so the next build will completely rebuild them
<code>make px4-v2-upload</code>	Build and upload the Pixhawk firmware for a quad (i.e. no need to do step #7 below)

The firmware will be created in the **ArduCopter** directory with the **.px4** file extension, ready to load onto the Pixhawk. For example if you build for px4-v2, **ArduCopter-v2.px4** will be created

7. Occasionally you should pull *PX4Firmware* and *PX4NuttX* updates. To make sure it compiles correctly, run the clean option in make:

```
make px4-clean
make px4-[frame type]
```

The available frame types are: quad, tri, hexa, y6, octa, octa-quad, hell.

8. To make and upload to your vehicle do:

```
make px4-quad-upload
```

## Building for Pixhawk/PX4 using Eclipse on Windows

This article shows how you can set up Eclipse for editing ArduPilot code and building for Pixhawk/PX4 targets.

### Note

Ensure that you have Java installed before Eclipse can run. If not, Java can be installed from:

<https://www.java.com/en/>

### Preconditions

Follow the instructions in [Building for Pixhawk/PX4 on Windows with Make](#) to download the required source code (*ardupilot*, *PX4Firmware* and *PX4NuttX*) and toolchain.

The *PX4 toolchain* includes a preconfigured version of Eclipse that has been set up for ArduPilot development.

### Starting Eclipse

Start Eclipse using the *PX4 Eclipse* link installed with the *PX4 toolchain*. The link can be accessed from either:

- The Windows Start menu (**Start | All Programs | PX4 Toolchain | PX4 Eclipse**), or
- You can directly run the file **C:\Pixhawk\_Toolchain\toolchain\msys\1.0\px4\_eclipse.bat**

### Creating the Project

You can set up an Eclipse project to build ArduPilot either “from scratch” or using pre-defined project files. Using the template files saves you a little work as they already include the project location and *px4-v2 make target definitions* for Copter and Plane.

#### Creating the Project from template files

The project can be created from predefined template files:

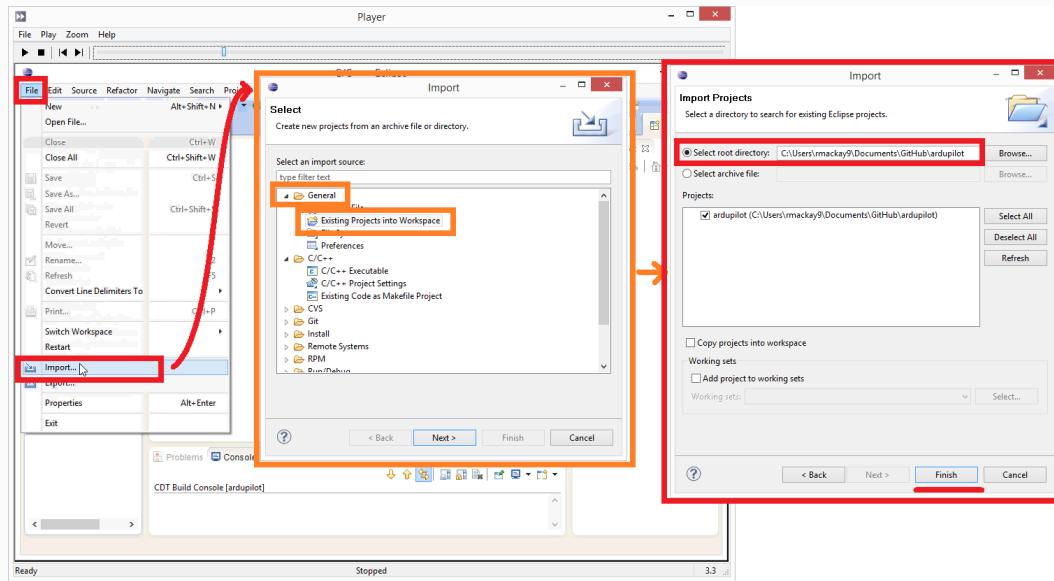
- Rename **/ardupilot/eclipse.cproject** to **.cproject** and **/ardupilot/eclipse.project** to **.project**

### Note

#### If using *Windows Explorer* append an additional period “.” to

the end of the files when renaming them - e.g. **.cproject**. (the additional period is not actually “saved”.)

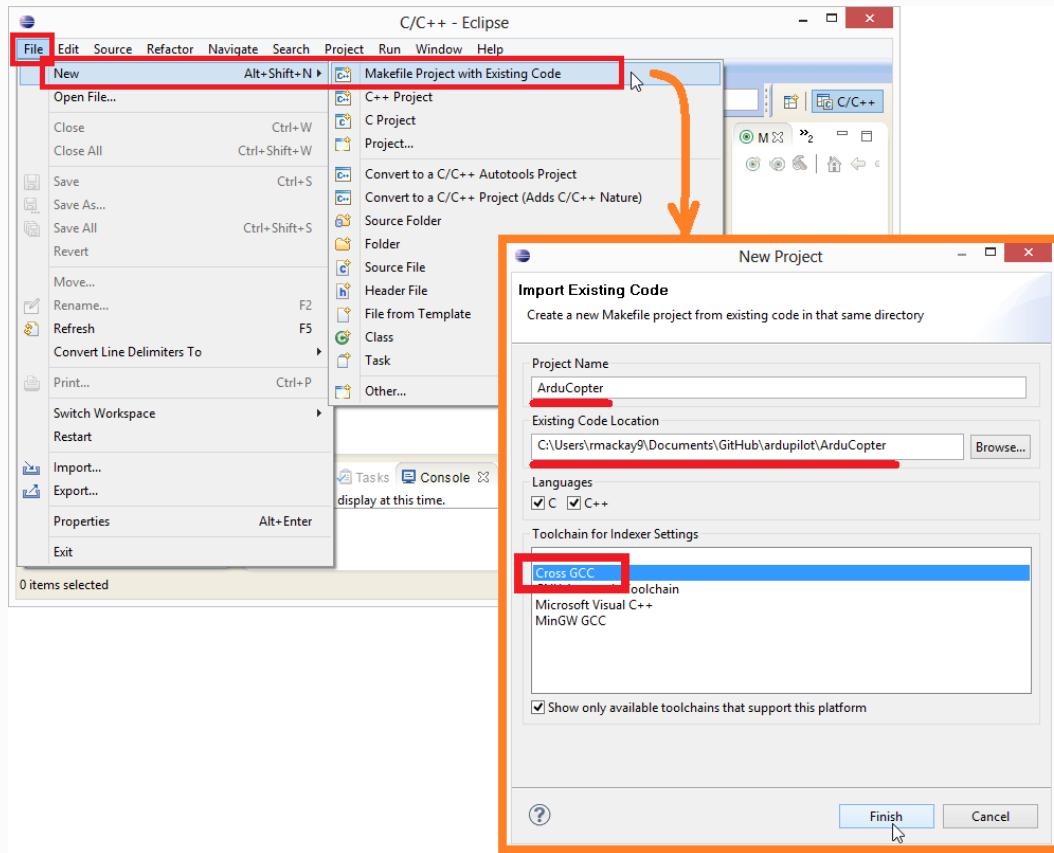
- Select **File | Import | General | Existing Projects into Workspace**
- Check **Select root directory** and browse to the ardupilot directory
- Select the ardupilot directory and press **Finish**



## Creating the Project from scratch

Alternative to the above steps, the project can be created by doing the following:

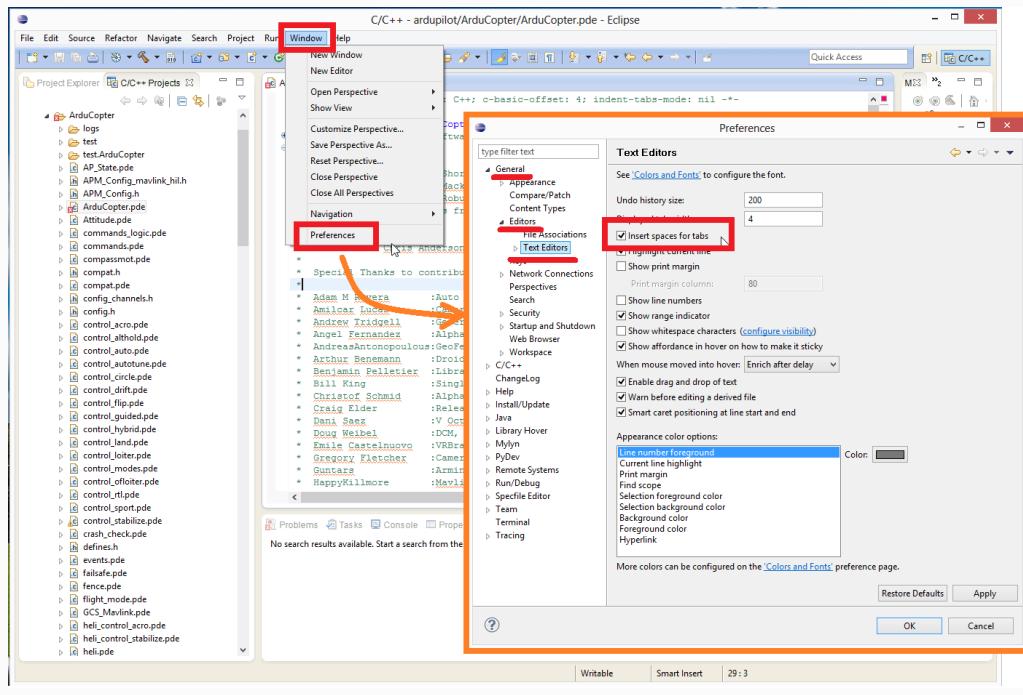
- Select **File | New | Make Project with Existing Code**
- Fill in the Project Name and set the *Existing Code Location* to the Copter directory
- Set the Toolchain to be *Cross GCC* and press **Finish**



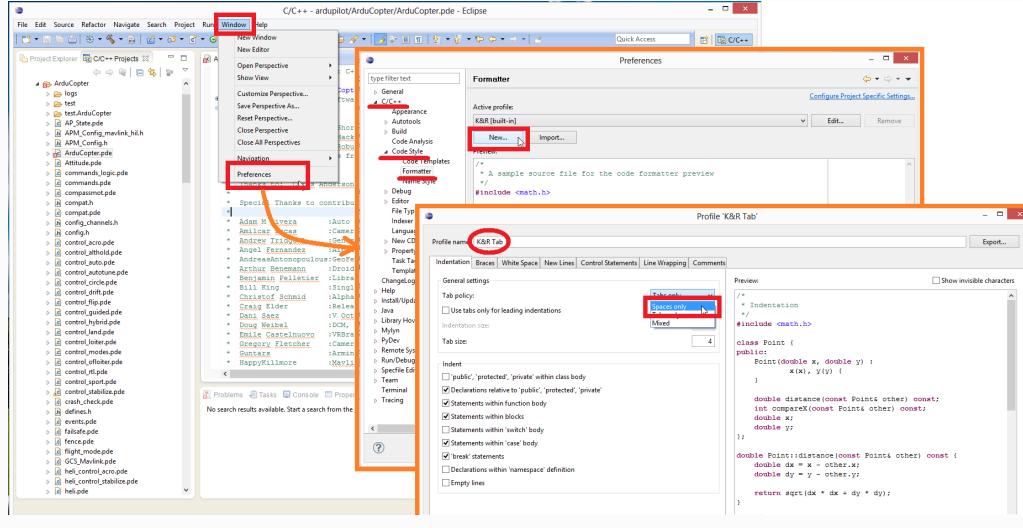
## Use spaces instead of tabs

By default Copter, Plane and Rover use spaces in place of tabs. This can be set to the default in Eclipse by changing two settings:

- Select Window | Preferences | General | Editors | Text Editors | Insert spaces for tabs.

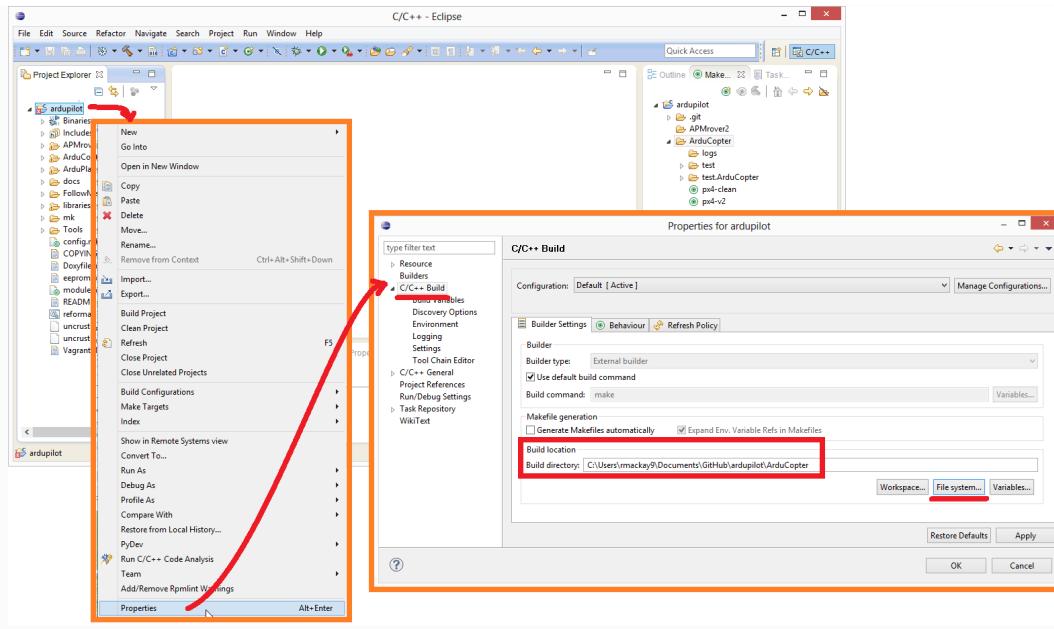


- Select Windows | Preferences | C/C++ | Code Style | Formatter and creating a new Profile (i.e. "K&R Tab") which has the "Indentation" set to "Spaces only"



## Specify build location

In the *Project Explorer* right-mouse-click on the ardupilot folder and select **Properties**. Then under C/C++ Build set the “Build location” to the Copter or Plane directory as shown below.

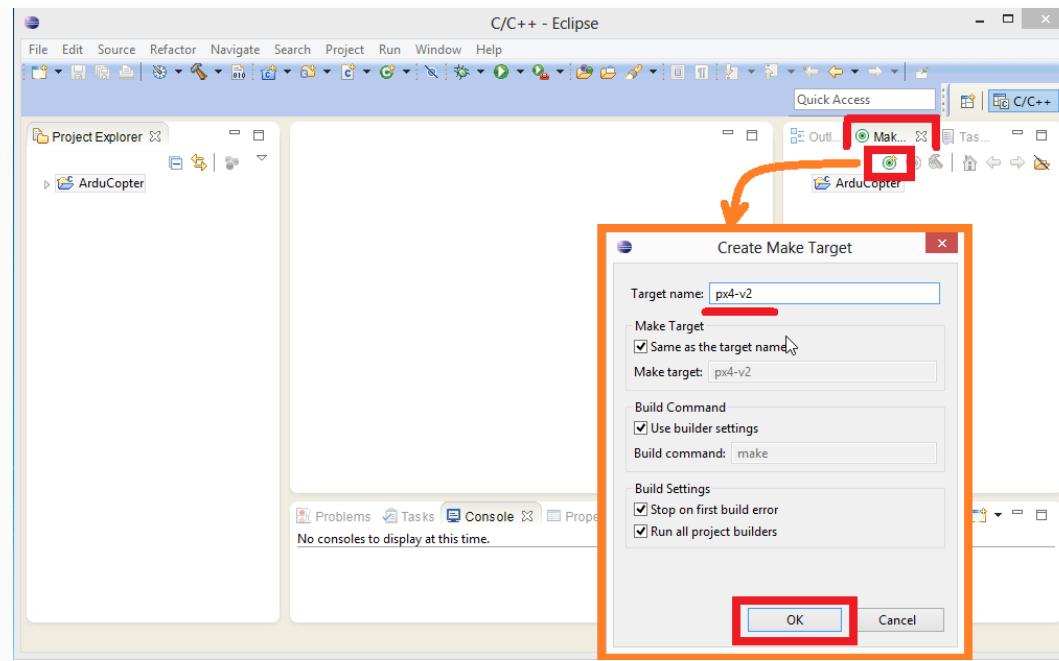


## Specify make targets

In the Make window on the right create any of these make targets (the full list of possible targets can be found in [px4\\_targets.mk](#)):

<code>make px4</code>	Build both PX4 and PixHawk firmware for a quadcopter
<code>make px4-v2</code>	Build the Pixhawk firmware for a quad
<code>make px4-v2-hexa</code>	Build the Pixhawk firmware for a hexacopter. # Other supported suffixes include "octa", "tri" and "heli". # More can be found in "mk/targets.mk" under FRAMES
<code>make clean</code>	"clean" the ardupilot directory
<code>make px4-clean</code>	"clean" the PX4Firmware and PX4NuttX directories so the next build will completely rebuild them
<code>make px4-v2-upload</code>	Build and upload the Pixhawk firmware for a quad (i.e. no need to do step #7 below)

For example, the image below shows how you might define a `px4-v2` make target.

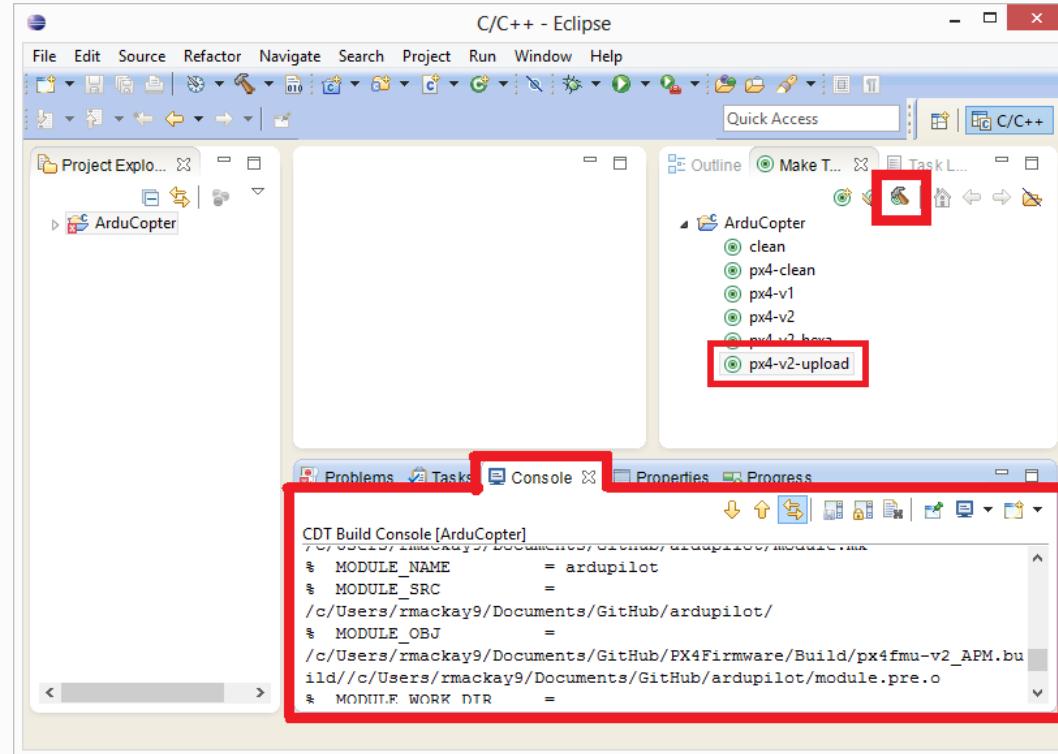


#### Note

There is currently no option to upload frames other than quad.

#### Building from Eclipse

The make target can be built by pushing the green circle + hammer icon. The build progress will appear in the Console window.



The firmware will be created in the vehicle directory (e.g. ArduCopter) and have the file extension .px4.

#### Building ArduPilot for Pixhawk/PX4 on Windows or Linux with QtCreator

This article shows how you can set up Qt Creator for editing ArduPilot code and building for Pixhawk/PX4 targets on Windows and Linux.

## Preconditions for Linux

Follow the instructions in [Building for Pixhawk/PX4 on Linux with Make](#) to download the required source code (*ardupilot*, *PX4Firmware* and *PX4NuttX*) and toolchain.

## Preconditions for Windows

Follow the instructions in [Pixhawk/PX4 on Windows with Make](#) to download the required source code (*ardupilot*, *PX4Firmware* and *PX4NuttX*) and toolchain.

Make sure you have no Cygwin installed (or have it at least out of the environment variables), as this can get in the way of the PX4 toolchain.

## Install Qt Creator

1. Download Qt and the Qt Creator IDE from the [Qt Creator website](#)
2. Follow the instructions to install the IDE on your platform.

## Run Qt Creator on Windows

We try to make sure that Qt Creator is running with all the necessary environment variables that are needed to build an ArduPilot project.

1. Go to your local PX4 toolchain directory and create a file called **qtcreator.sh**.

```
cd /path-to-your-qt-creator-dir/bin
qtcreator.exe
```

2. Go to the *toolchain\msys\1.0* subdirectory of the PX4 toolchain directory and make a copy of the file **px4\_console.bat**, called **px4\_qt\_creator.bat**. Change this file in *:startsh* section, so that it becomes:

```
:startsh
if NOT EXIST %WD%sh.exe goto notfound
start %WD%sh --login -i -c qtcreator.sh
exit
```

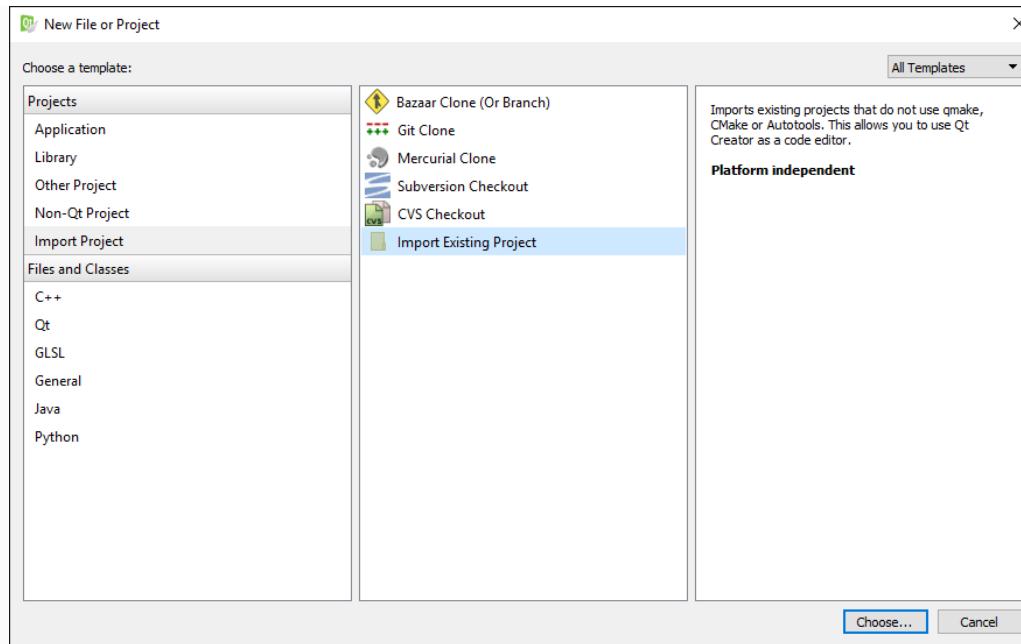
3. Optional: create a Windows shortcut to the **px4\_qt\_creator.bat** for easy access.
4. Start up Qt Creator by starting up **px4\_qt\_creator.bat**

## Run Qt Creator on Linux

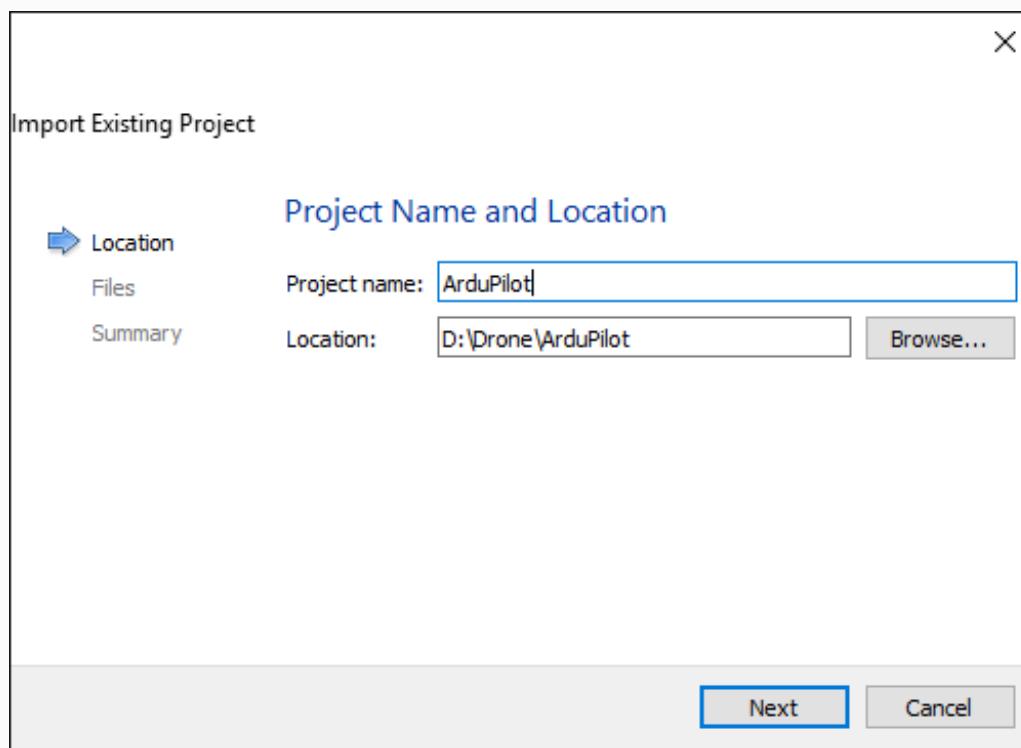
If you installed the gcc-arm cross-compiler and made sure that the cross-compiler is in your path, then it suffices to simply start up Qt Creator.

## Create a project

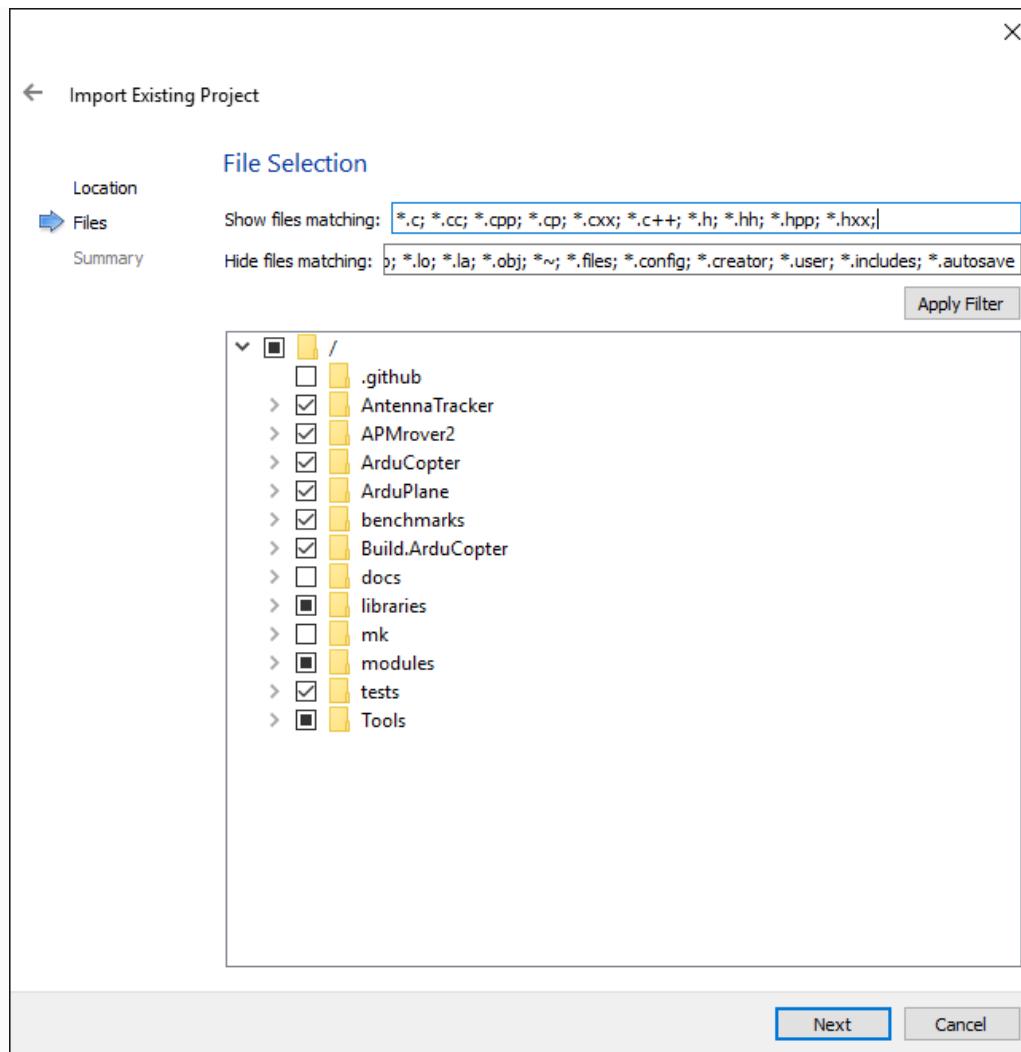
1. Select **File -> New File or Project**.
2. Choose the **Import Project** template and from these templates **Import Existing Project**. Then press **Next**.



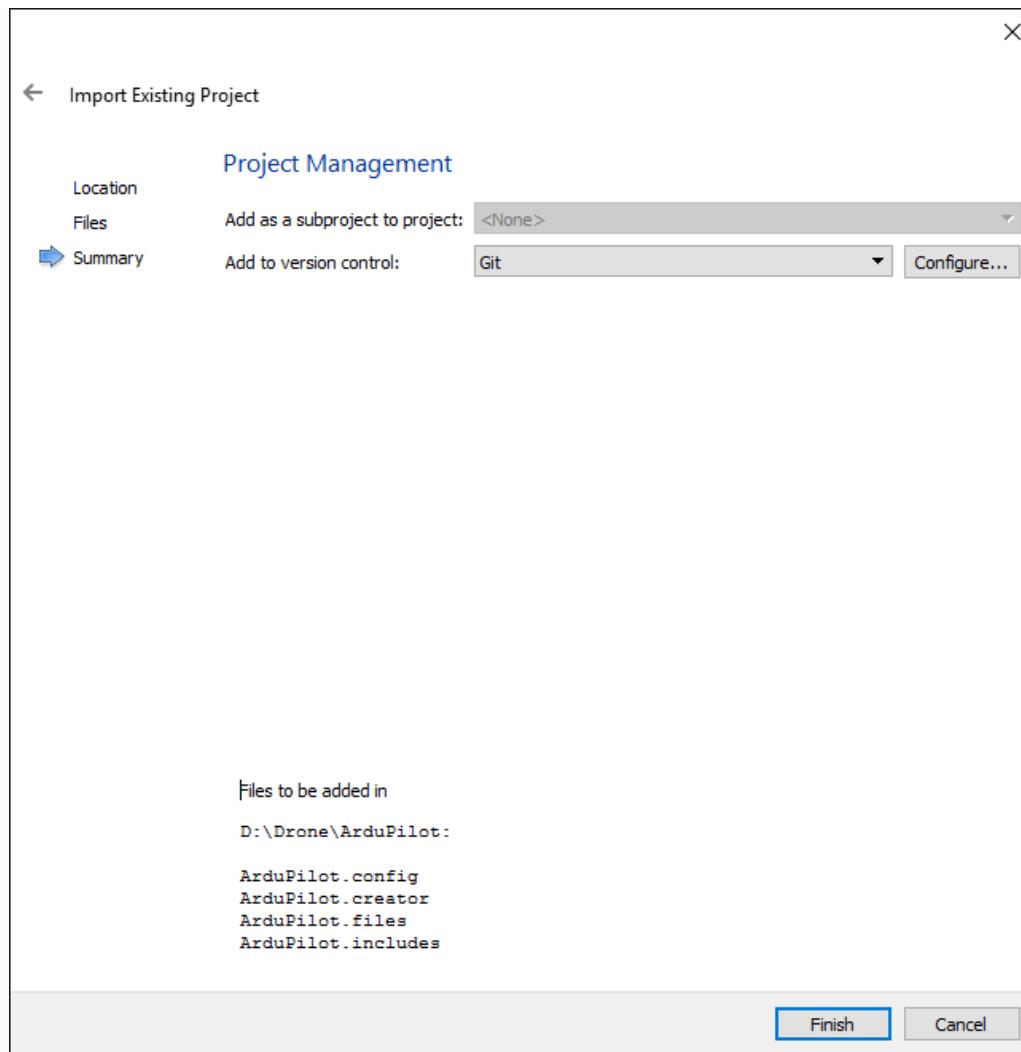
3. Enter a project name and choose the location of the ArduPilot Git repository. Then press **Next**.



4. Qt Creator shows you the files that will be imported into the project. Just press **Next** (we will worry about this a bit later).



5. The summary of the project settings is shown in the next screen. It is interesting to see which files are generated:
  1. the **.files** file contains all the files that need to be edited.
  2. the **.includes** file contains all directories containing header files that might be useful to consult during development.



## Update the project with Git hooks

A fixed project is not useful, because files can get renamed or be added or removed by commits from other contributors.

Therefore, it is useful to let the Qt Creator project be updated each time a new incoming change from a remote repository updates your own repository.

The tactic is that we first create a “project generation script” that will update the project’s **.files** and **.includes** files and then let Git hooks call this script each time when it assumed to be appropriate.

Windows script [¶](#)

Create a file called **generate\_ardupilot\_project.bat**:

```
@echo off
cd ArduCopter
dir *.cpp *.hpp *.ipp *.c *.h /b /s > ..\ArduPilot.files
cd ..
cd AntennaTracker
dir *.cpp *.hpp *.ipp *.c *.h /b /s >> ..\ArduPilot.files
cd ..
cd ArduPlane
dir *.cpp *.hpp *.ipp *.c *.h /b /s >> ..\ArduPilot.files
cd ..
cd APMRover2
dir *.cpp *.hpp *.ipp *.c *.h /b /s >> ..\ArduPilot.files
cd ..
dir *include* /A:D /s /b > ArduPilot.includes
```

```
dir *libraries /A:D /s /b >> ArduPilot.includes
```

## Linux script¶

Create a file called **generate\_ardupilot\_project.sh**:

```
cd ArduCopter
find . \( -name "*.cpp" -o -name "*.hpp" -o -name "*.ipp" -o -name "*.c" -o -name "*.h" \) >
../ArduPilot.files
cd ..
cd AntennaTracker
find . \( -name "*.cpp" -o -name "*.hpp" -o -name "*.ipp" -o -name "*.c" -o -name "*.h" \) >>
../ArduPilot.files
cd ..
cd ArduPlane
find . \( -name "*.cpp" -o -name "*.hpp" -o -name "*.ipp" -o -name "*.c" -o -name "*.h" \) >>
../ArduPilot.files
cd ..
cd APMRover2
find . \( -name "*.cpp" -o -name "*.hpp" -o -name "*.ipp" -o -name "*.c" -o -name "*.h" \) >>
../ArduPilot.files
cd ..
find . -type d -name 'include' > ArduPilot.includes
find . -type d -name 'libraries' >> ArduPilot.includes
```

## Git hooks¶

Open a command line interface and browse to the **.git/hooks** subfolder in the project folder.

Change the **post-merge** and **post-checkout** files so that they become:

```
#!/bin/sh
./generate_qt_creator_files.bat
exit 0
```

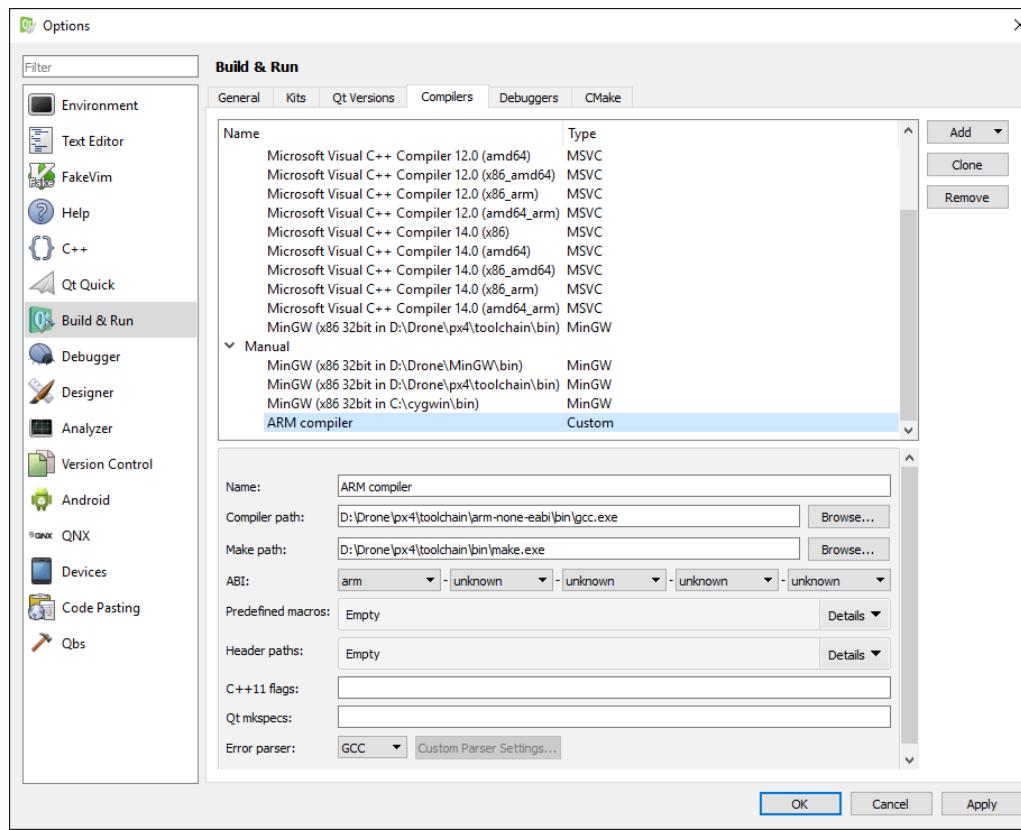
Another option is to make symbolic links in between the Git hook files and the generation script. In Linux for example, that is achieved by:

```
ln -s ./generate_ardupilot_project.sh ./git/hooks/post-merge
ln -s ./generate_ardupilot_project.sh ./git/hooks/post-checkout
```

## Build the project

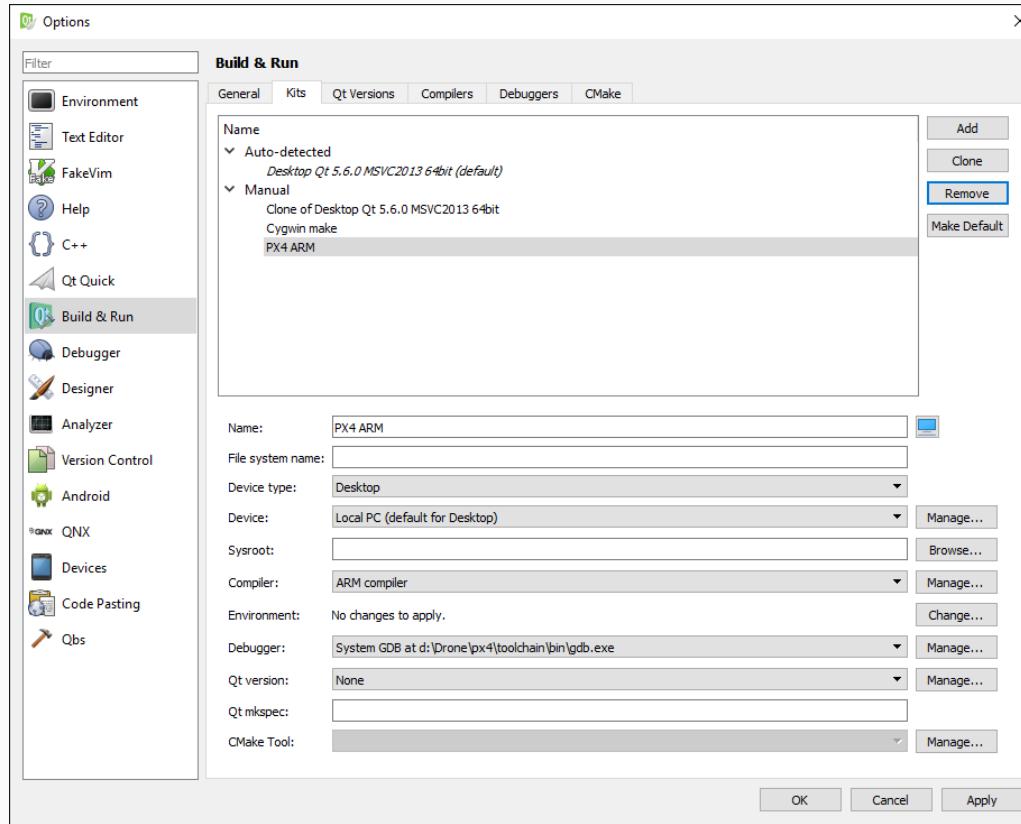
This section discusses how to build the code in Qt Creator.

1. Click on **Projects** on the left pane and make sure that you are in the **Build & Run** tab page.
2. Click **Manage Kits** in the topleft corner.
3. First click on the **Compilers** tab page and then **Add** on the right hand side of the compilers list.  
Choose an easily recognisable name for your compiler and make sure the Compiler and Make path are referring to the executables of the PX4 toolchain (Windows) or the downloaded gcc-arm cross-compiler (Linux). Also choose “GCC” as the Error parser.



- Then click on the **Kits** tab page. Click **Add** on the right hand side.

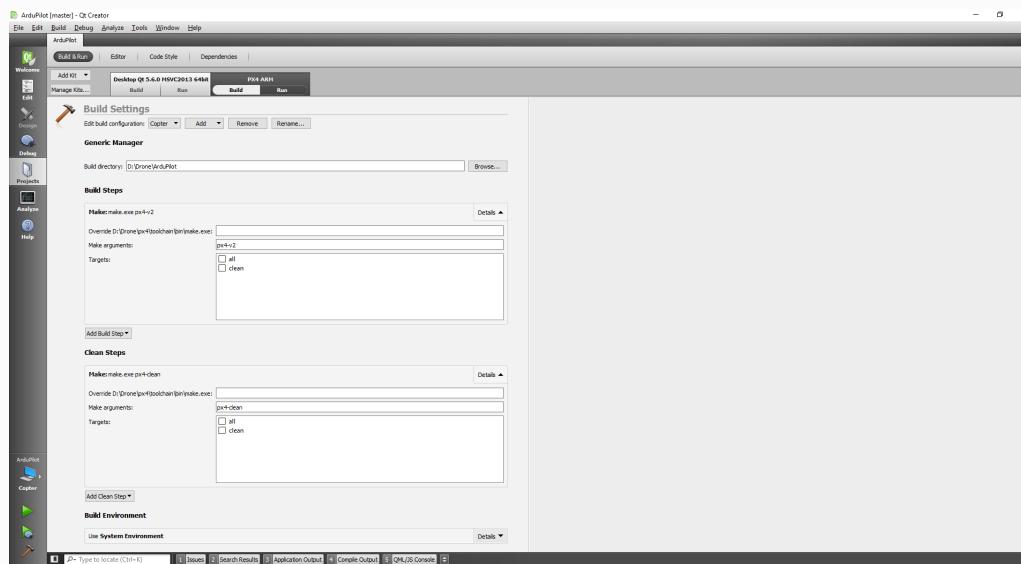
Choose an easily recognisable name for your build kit and make sure you fill in the proper compiler (the one you just added) and the debugger inside the PX4 toolchain.



- Click **Apply**.
- Back on the **Projects** page, click **Add Kit** and choose the Build Kit you just added.
- Now you have one "Build Configuration" called "Default". You can make as many Build Configurations as you want, but we'll take the ArduCopter build as an example for now. Next to *Edit build*

configuration, click on **Add** and choose **Clone Selected** in the drop down menu. Pick a name (e.g. "Copter").

8. Click on the **Details** of the *Build Steps* and type "px4-v2 -j2" as *Make arguments*. Deselect the **Targets**.
9. Click on the **Details** of the *Clean Steps* and type "px4-clean" as *Make arguments*. Deselect the **Targets**.



10. You can make other build configurations for e.g. ArduPlane in the same way. You can quickly switch between "Build Configurations" by clicking the logo just above the **Run** icon (the green arrow) on the left pane.
11. You can now remove the MSVC or standard GCC build kit (click on the down arrow on the kit itself and choose **Remove Kit**).
12. You're now ready to build the code. Click on **Edit** in the left pane to edit the code and browse through the project. Click **Build project-name** in the *Build* menu (or **Ctrl+B**) to build the code.

## Apply coding style guidelines

It is useful that the Qt Creator editor is configured so that it automatically applies the layout guidelines described in [ArduPilot Style Guide](#).

1. Indentation: Click on the **Tools** menu and choose **Options**. Subsequently, pick the **Text Editor** view and then the **Behaviour** tab page. You can set the tab policy (spaces only) and the size of a tab and indentations (4).
2. Other interesting settings can be found in the **C++** view in the same *Options* dialog. You can define how specific parts of your code will be aligned (e.g. assignments, switch/cases, control statements, braces, etc.)
3. Commenting: In order to comply with the coding guidelines , you will need to provide documentation in Doxygen format. Qt Creator will automatically generate a Doxygen documentation template if you type `/*` before the definition of the class, function, ...

## Building ArduPilot for APM2.x with Make

This article explains how to build the code for APM2.x with Make on Windows, Mac and Linux.

### Warning

Copter 3.3 firmware (and later) and builds after Plane 3.4.0 no longer fit on APM boards. Plane, Rover

and AntennaTracker builds can still be installed at time of writing but you can no longer build APM2.x off the master branch (you will need to build off a supported release branch).

The last Copter firmware that can be built on APM 2.x [can be downloaded from here](#).

## Overview

The preferred way for developers to build APM applications is using the Makefile system distributed with the repository. This Makefile system is the canonical way to build APM. APM applications can no longer be built with the standard Arduino IDE, as the APM build requires the Arduino IDE base libraries to be excluded from the build.

For developers who would prefer not to use make, a modified Arduino IDE which emulates the custom APM build process is available.

## Requirements

The APM makefiles will work out of the box for Mac OS and Linux users. Windows users can use the makefiles via a Unix compatibility system such as Mingw or Cygwin.

### Mac OS

Mac OS system requirements are:

- An Arduino install of version 1.0 or greater. The Arduino installation provides an AVR compiler.
- Apple Developer Tools (a free download from Apple). These provide command line utilities including GNU Make.

Arduino should be installed in a location that is indexed by Spotlight.

### Linux

Follow the instructions on [this page](#) to build the code with Linux. Linux system requirements vary depending on the distribution. The following are essential:

- The GNU sed utility.
- The GNU make utility, sometimes referred to as 'gmake'.
- The GNU awk utility, often referred to as 'gawk'. **Warning:** Many Linux distributions do not come with 'gawk' by default. You can test for the presence of this tool with the 'which gawk' command.
- The AVR-GCC toolchain.

There are various ways to get all of these utilities for your system. You may consult [Arduino's Linux guide](#) for instructions on installing the AVR toolchain under Linux.

### Windows

Follow the instructions on [Building ArduPilot for APM2.x on Windows with Make](#). These instructions make use of the *PX4 Toolchain*'s development environment.

## Preparing

Locally clone the APM repository from <https://github.com/ArduPilot/ardupilot>

```
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

## Building

To build a sketch for the default hardware platform you specified in **config.mk**, you can invoke *make* with no arguments in a sketch directory:

```
$ cd ArduCopter
$ make
```

The output of the build is, by default, located in **\$TMPDIR/\_sketchname\_.build**

```
$ make
% param_table.o
% ArduCopter.cpp
% ArduCopter.o
...
% ArduCopter.elf
% ArduCopter.eep
% ArduCopter.hex
```

## Uploading

If you have configured the `PORT` variable properly, the `upload` target will use `avrdude` to upload a built APM application to AVR based platforms.

```
$ cd ArduPlane
$ make upload
```

For PX4 platforms, the `px4-upload` target will use the PX4 bootloader to perform an upload.

## Troubleshooting

The build process itself attempts to diagnose problems that would prevent a successful build outside of code errors. It may emit the following diagnostics:

### **WARNING: More than one copy of Arduino was found, using ... (Mac OS only)**

Spotlight found more than one copy of Arduino installed on your system. Check the path that is printed to ensure that the correct version is being used. To avoid this problem, either remove old versions of Arduino or specify the ARDUINO option when invoking the build system. Typically the installation that will be chosen is the one that was most recently launched.

### **ERROR: must set BOARD before including this file.**

The sketch Makefile has not defined the BOARD variable. This is normally set to `atmega2560` for APM2.x builds.

### **ERROR: Spotlight cannot find Arduino on your system. (Mac OS only)**

Arduino is not installed, or it is installed in a location that is not being indexed by Spotlight. You can either enable Spotlight for the location where Arduino is installed, or specify the location explicitly by setting the ARDUINO option as described above. Note that Spotlight indexing may take some time, so enabling it for the location containing Arduino may not immediately correct this issue.

### **ERROR: Cannot find Arduino on this system**

(Linux and Windows only)Arduino was not found in one of the standard locations. Either move Arduino to a standard location, or specify its current location with the ARDUINO option in the **config.mk** file

### **ERROR: cannot find the compiler tools anywhere on the path ...**

The compiler and related tools cannot be found. For Mac OS and Windows the tools are normally part of Arduino and this message indicates that the Arduino installation is damaged.

For Linux systems, this means that the AVR tools are not installed in a standard location. Either set the TOOLPATH option to point to the directory containing the AVR tools, or install them in a standard location. Normally installing Arduino on a Linux system will result in a correct installation of the AVR tools.

### **ERROR: cannot find gawk - you may need to install GNU awk**

(Linux and Windows only)The GNU awk utility is required, but it has not been installed or cannot be found. Check that `gawk –version` works at the command prompt. You may need to specify its location explicitly with the AWK option.

Building ArduPilot for APM2.x on Linux with Make¶

Quick start¶

For Ubuntu, follow these steps to build the code. For other distributions, see the advanced instructions below.

Setup¶

Install git:

```
sudo apt-get -qq -y install git
```

Clone the source:

```
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

Run the install-prereqs-ubuntu.sh script:

```
ardupilot/Tools/scripts/install-prereqs-ubuntu.sh -y
```

Reload the path (log-out and log-in to make permanent):

```
. ~/.profile
```

Build¶

Build for Copter:

```
cd ardupilot/ArduCopter
make
```

Build for Plane:

```
cd ardupilot/ArduPlane
make
```

Build for Rover:

```
cd ardupilot/APMrover2
make
```

Build for Antenna Tracker:

```
cd ardupilot/AntennaTracker
make
```

#### Advanced¶

To build the ardupilot firmware on Linux, you will need a cross-compiler for the type of board you are building for. If you are using a debian based distribution such as Ubuntu then the following should get you the core packages you will need:

```
sudo apt-get install gcc-avr avrdude avr-libc binutils-avr
```

You will also need some extra tools to use the SITL system and additional development tools

```
sudo apt-get install python-serial python-wxgtk2.8 python-matplotlib python-opencv python-pexpect
python-scipy
```

Once installed, you can build by changing directory to the vehicle type you want to build for, and running ‘make’ with the target you want. Look in mk/targets.mk for a list of build targets.

#### Ubuntu Linux¶

The following packages are required to build ardupilot for the APM1/APM2 (Arduino) platform in Ubuntu:

```
gawk make git arduino-core g++
```

To build ardupilot for the PX4 platform, you’ll first need to install the PX4 toolchain and download the PX4 source code. See the [PX4 toolchain installation page](#).

The easiest way to install all these prerequisites is to run the [ardupilot/Tools/scripts/install-prereqs-ubuntu.sh](#) script, which will

install all the required packages and download all the required

software.

#### Building using make¶

1. For Copter and AntennaTracker you’ll need to run `make configure` from a sketch directory (ArduCopter or AntennaTracker) before you build the project for the first time. This will create a **config.mk** file at the top level of the repository. You can set some defaults in **config.mk**

2. In the sketch directory, type `make` to build for APM2. Alternatively, `make apm1` will build for the APM1 and `make px4` will build for the PX4. The binaries will generated in `/tmp/sketchname.build`.

3. Type `make upload` to upload. You may need to set the correct default serial port in your `config.mk`.

#### Building ArduPilot for APM2.x on Windows with Make¶

This article shows how to build ArduPilot for APM2.x on Windows with *Make*.

Tip

The approach described here is useful if you want to [develop using Eclipse](#). [Building ArduPilot for APM2.x on Windows with Arduino](#) shows an alternative method for building ArduPilot for APM2.x.

#### Warning

Copter 3.3 firmware (and later) and builds after Plane 3.4.0 no longer fit on APM boards. Plane, Rover and AntennaTracker builds can still be installed at time of writing but you can no longer build APM2.x off the master branch (you will need to build off a supported release branch, or for the keen developer, from the AVR-master branch master-AVR and the tags from there. see:

<https://github.com/ArduPilot/ardupilot/tree/master-AVR> ).

The last Copter firmware that can be built on APM 2.x [can be downloaded from here](#).

#### Overview

These instructions use the [PX4 Toolchain](#) along with the [Arduino Tools](#) to set up an environment in which you can build for APM2.x targets with *make*. They have been tested on Windows 10 to build the ArduCopter-3.2.1 branch.

#### Note

This article replaces previous instructions to use a basic [Cygwin](#) installation with the GNU sed, make and awk packages installed. The pre-built environment from the PX4 Toolchain is a lot easier to set up.

#### Build instructions

1. Install [GitHub for Windows](#)

2. Ensure your github settings are set to leave line endings untouched.

- The “Git Shell (or Bash)” terminal was also installed when you installed Git. Click on your new “Git Shell (or Bash)” Icon and type in the following in the Git “MINGW32” Terminal window:

```
git config --global core.autocrlf false
```

3. Get the source code onto your machine

- In the Git “MINGW32” Terminal window navigate to where you want to put the source code and clone the repo

```
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

- Checkout the branch you want to build (the last branch you can use for Copter is shown below):

```
git checkout ArduCopter-3.2.1
```

4. Install the special ArduPilot Arduino package. This contains gcc 4.8.2 and Eclipse “Luna”.

- Download the installation zip: [ArduPilot-Arduino-1.0.3-gcc-4.8.2-windows.zip](#)
- Unzip the file to the root of the C drive

#### Note

#### You can install

anywhere. Later on we update **config.mk** to tell the build system where the tools are located.

5. Download and install the *PX4 toolchain* by running the [px4\\_toolchain\\_installer\\_v14\\_win.exe](#)

6. Open the *PX4Console* and navigate to the target vehicle directory:

- Start the *PX4Console*. This can be found under **Start | All Programs | PX4 Toolchain** (Windows 7 machine) or you can directly run **C:\px4\toolchain\msys\1.0\px4\_console.bat**

Navigate to the vehicle-specific ArduPilot directory in the *PX4Console*. For example, to build Copter, navigate to:

```
cd /c/Users/<username>/Documents/GitHub/ardupilot/ArduCopter
```

## 7. Configure the build system to find the *Arduino tools*:

- Enter the following command on the *PX4Console* to create */ardupilot/config.mk*.

```
make configure
```

- Open **config.mk** (created in the directory above ArduCopter) and define the **ARDUINO** variable as shown:

```
ARDUINO = C:/arduino-1.0.3-windows
```

### Note

**You must specify the drive letter and use forward slashes for the path.**

## 8. Build the firmware by entering the following command on the *PX4Console*:

```
make apm2
```

### Tip

This command can take several minutes before it is obvious that something is happening! The firmware will be created in a subfolder of the user's temp directory. For example you will find **ArduCopter.hex** in **C:\Users\\*YourUserNameHere\*\AppData\Local\Temp\ArduCopter.build**.

## 9. Upload the firmware using the *Mission Planner Initial Setup | Install Firmware* screen's **Load custom firmware** link

### Hints for speeding up compile time

Anti virus protection is likely to slow the compile times especially for PX4 so it is recommended that the folders containing the ArduPilot source code is excluded from your virus protections real-time scan.

The first scan after a **make px4-clean** will be very slow as it rebuilds everything

## Building ArduPilot for APM2.x on Windows with Arduino

This article shows how to build ArduPilot for APM2.x targets on Windows, using the Arduino toolchain.

### Tip

An alternative approach is covered in [Building ArduPilot for APM2.x on Windows with Make](#).

### Warning

Copter 3.3 firmware (and later) and builds after Plane 3.4.0 no longer fit on APM boards. Plane, Rover and AntennaTracker builds can still be installed at time of writing but you can no longer build APM2.x off the master branch (you will need to build off a supported release branch).

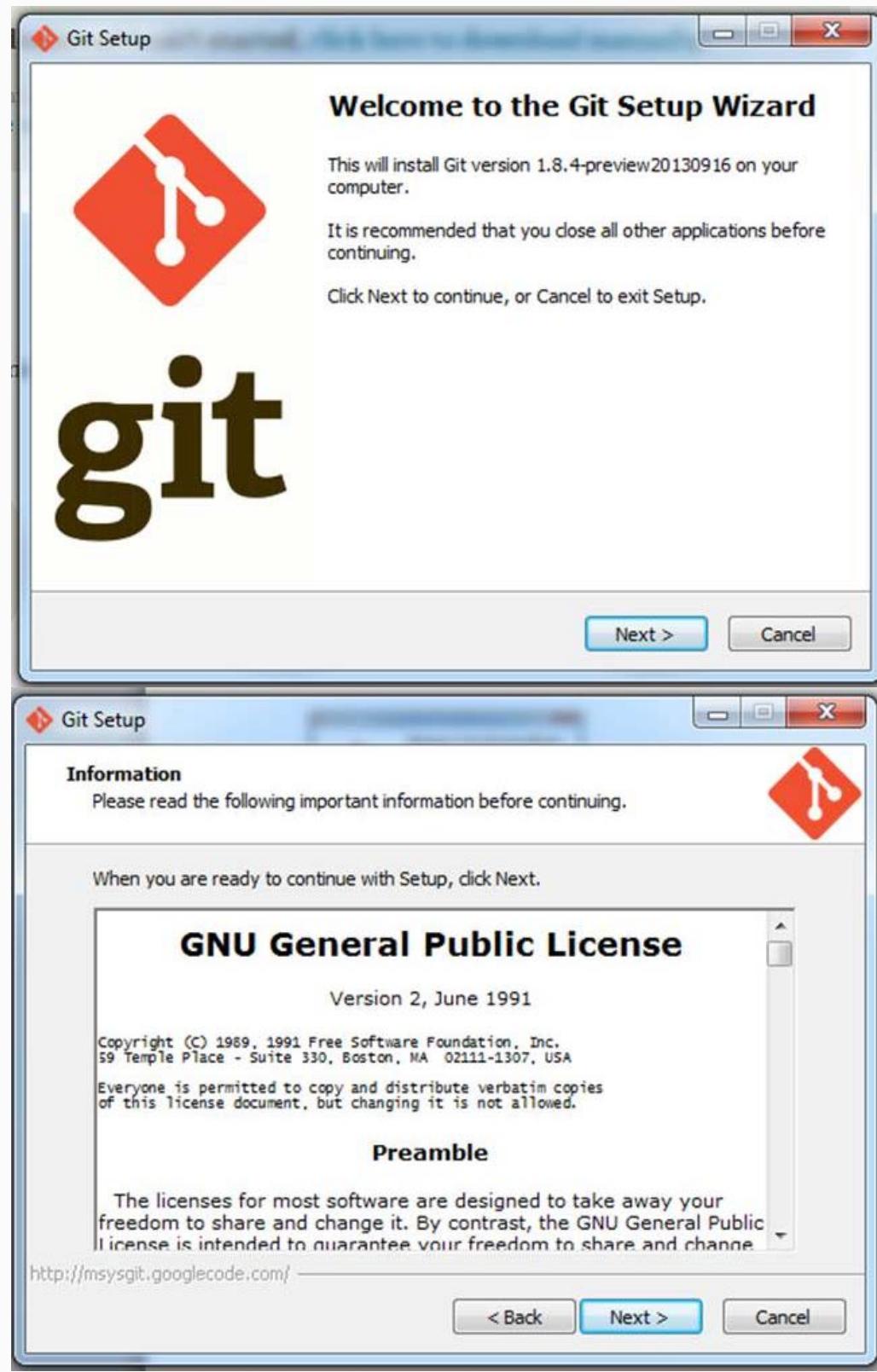
The last Copter firmware that can be built on APM 2.x [can be downloaded from here](#).

In addition to the above restrictions, this article covers:

- From Copter version 3.1 to version 3.2.1
- From Plane version 2.76 to version 3.4.0
- APM 2.0, 2.5, and 2.6 only

### Install Git-SCM

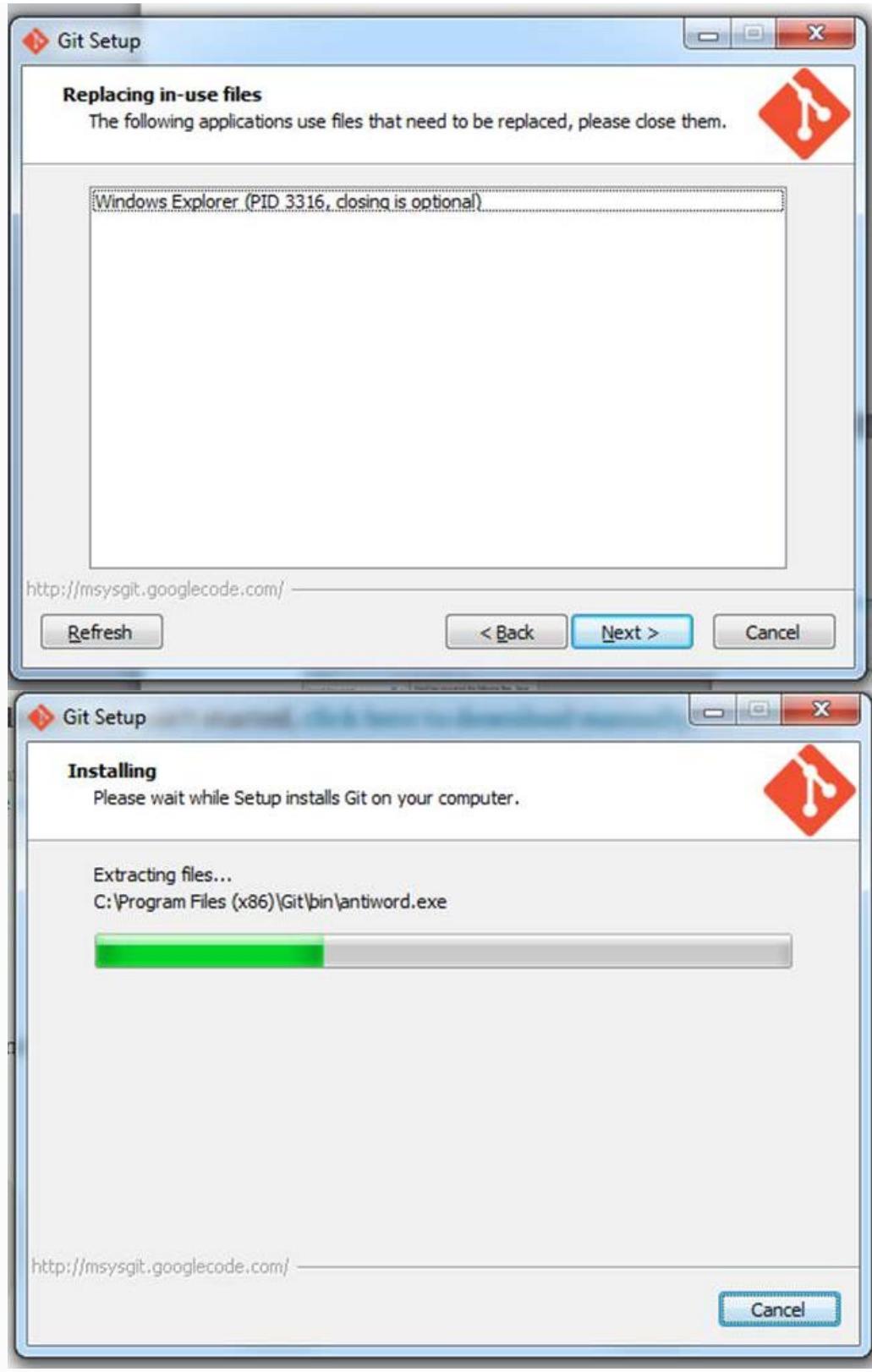
1. Download and run the install file from: <http://git-scm.com/download/win>
2. Follow the screenshots below to make your selections during install.



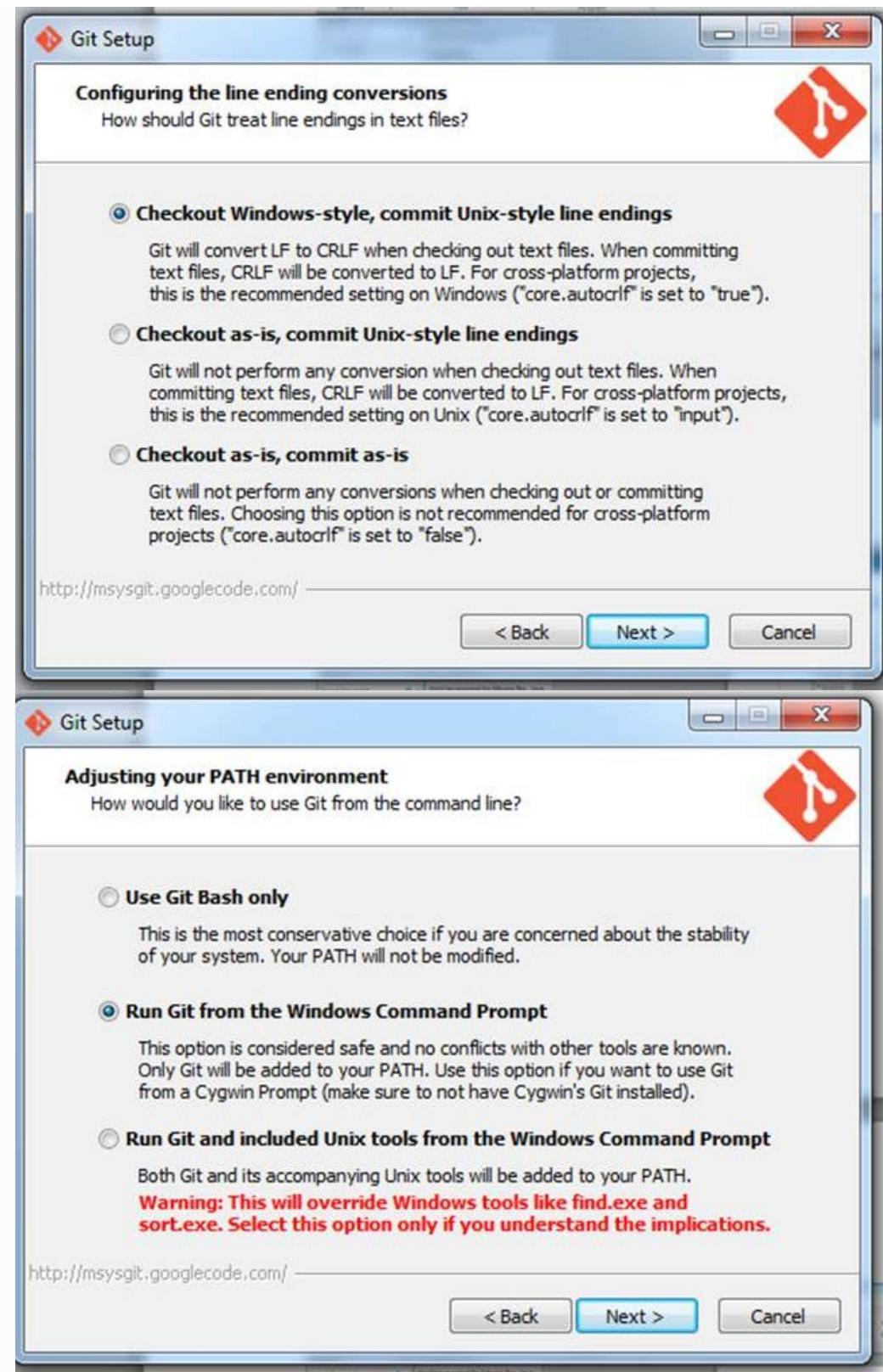
3. On the *Welcome* screen and then again on the *License* screen click the **Next** button



4. On the **Select Components** screen click on the **Next** button, then click the **Finish** button



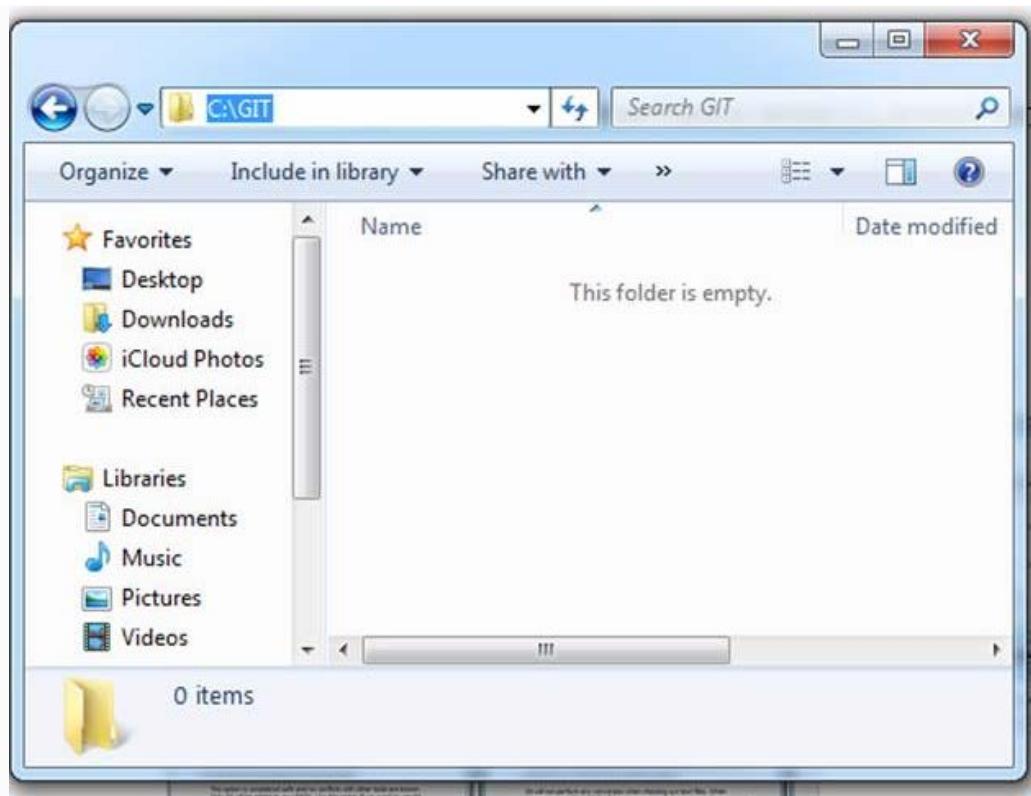
5. Click the **Next** button in the *Replacing in Use Files Screen*, then wait for Git to finish loading



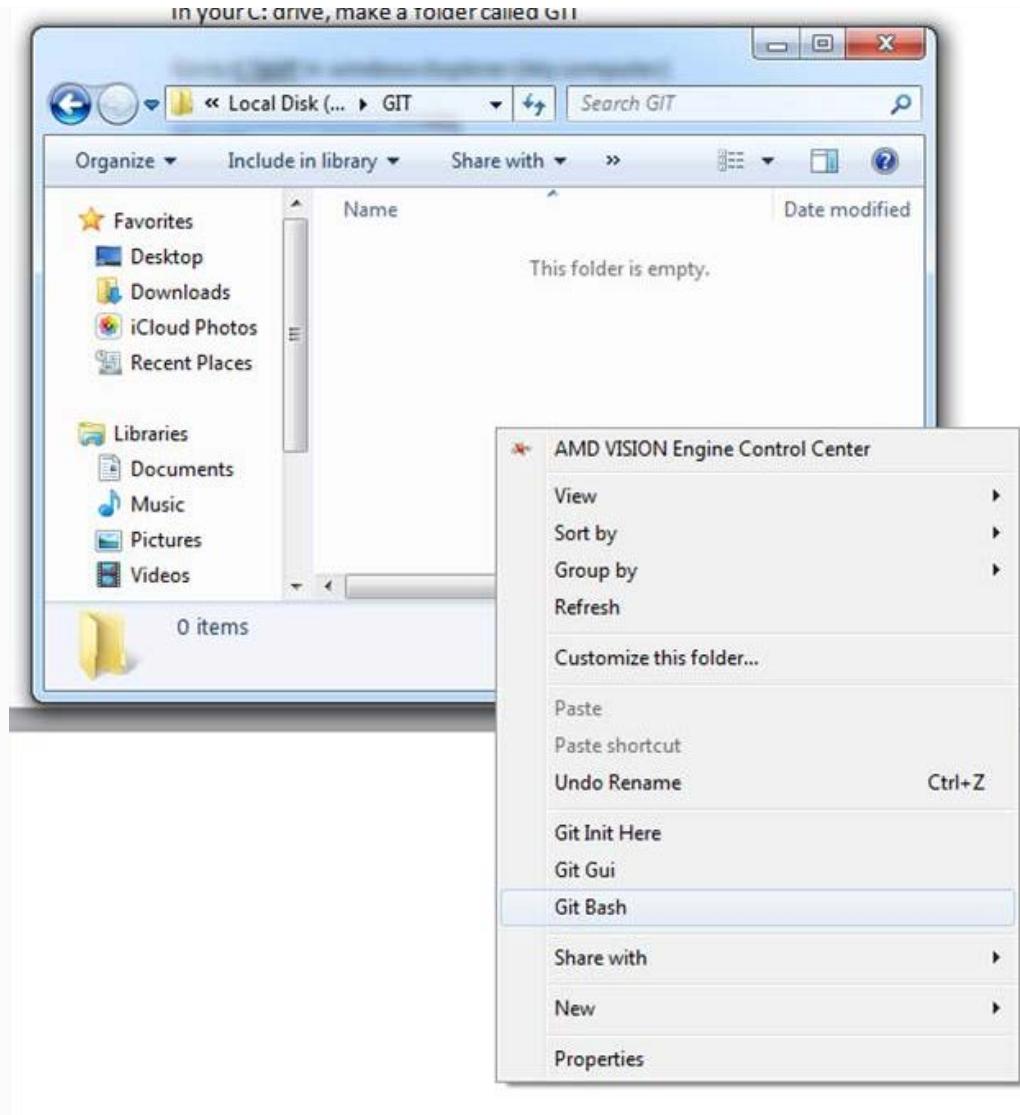
6. Select the **Checkout Windows** item and the **Next** button then Select the **Run Git from Windows** item and the **Next** button.

#### Download source

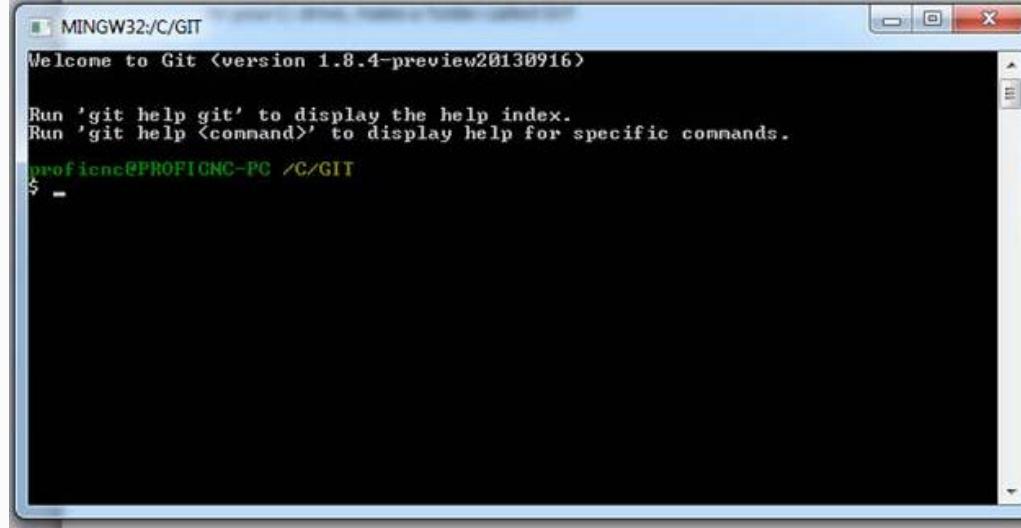
- In your C: drive, make a folder called GIT (C:\GIT on my computer). Navigate to the folder Windows Explorer



2. Right click anywhere in the folder and click git bash



This screen will come up



3. In this screen type

```
git clone git://github.com/ArduPilot/ardupilot.git
```

```
Welcome to Git <version 1.8.4-preview20130916>
Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
proficnc@PROFICNC-PC /C/GIT
$ git clone git://github.com/diydrones/ardupilot.git
```

When it is finished it should look like this....

```
Welcome to Git <version 1.8.4-preview20130916>
Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
proficnc@PROFICNC-PC /C/GIT
$ git clone git://github.com/diydrones/ardupilot.git
Cloning into 'ardupilot'...
remote: Counting objects: 74764, done.
remote: Compressing objects: 100% <20379/20379>, done.
remote: Total 74764 <delta 57833>, reused 69533 <delta 53044>
Receiving objects: 100% <74764/74764>, 40.27 MiB / 533.00 KiB/s, done.
Resolving deltas: 100% <57833/57833>, done.
Checking connectivity... done
Checking out files: 100% <1566/1566>, done.
proficnc@PROFICNC-PC /C/GIT
$ -
```

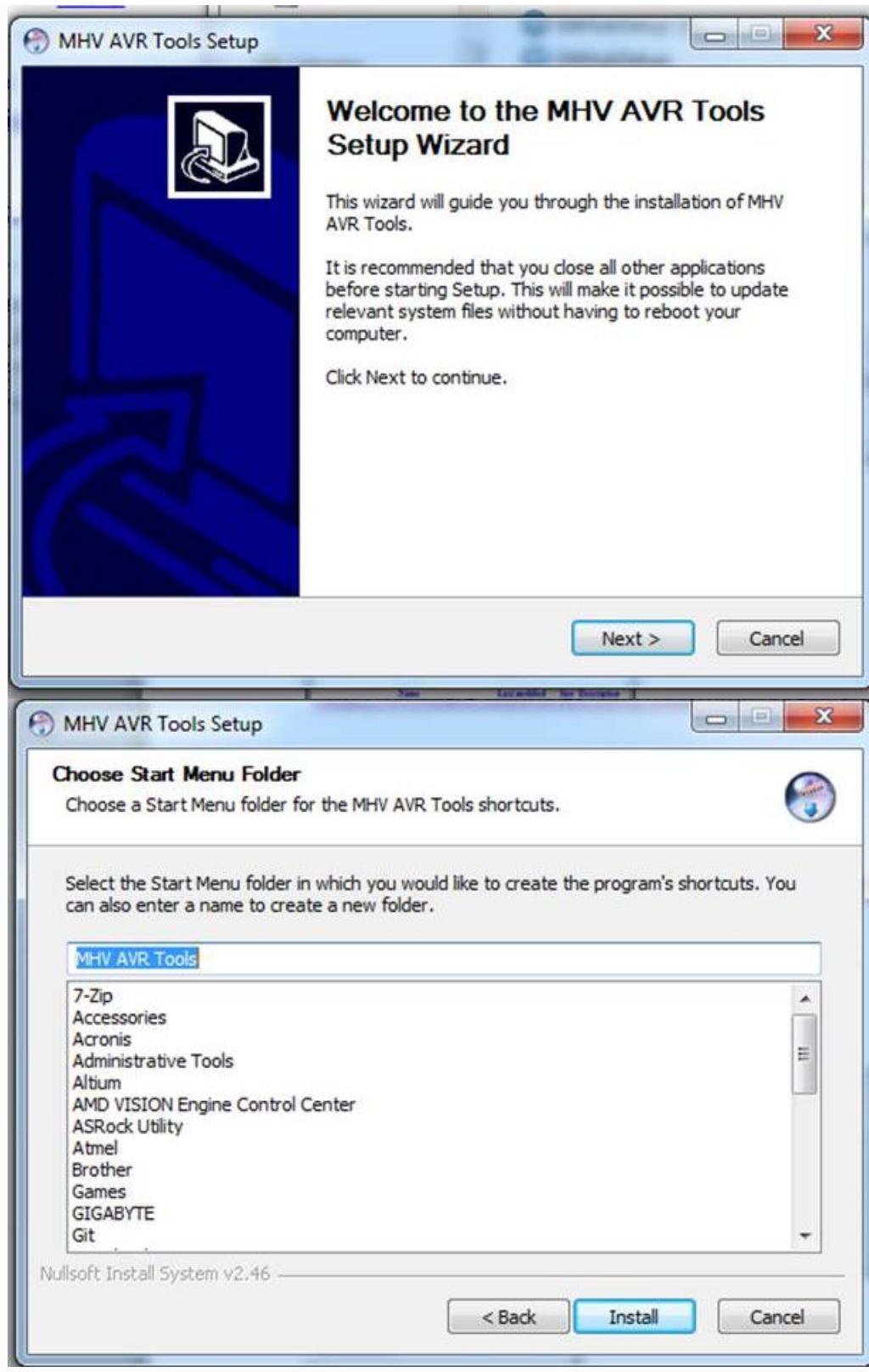
4. A little more initialisation is required for the source code. Initialise referenced dependencies like this:

```
cd ardupilot
git submodule update --init --recursive
```

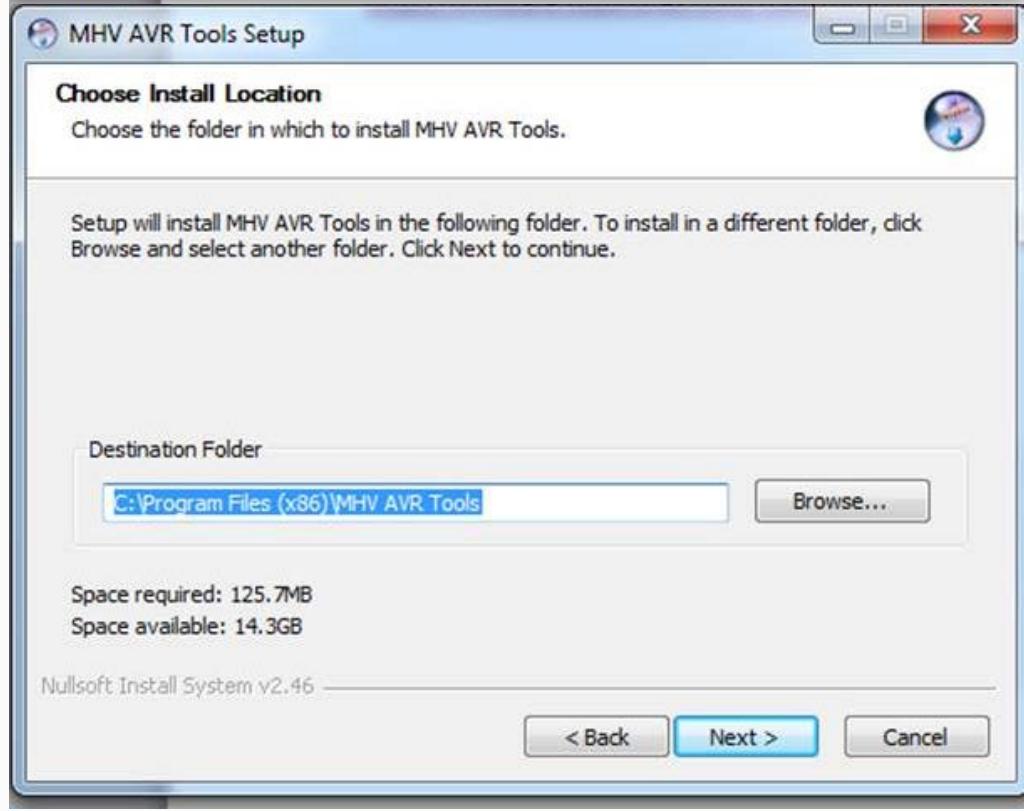
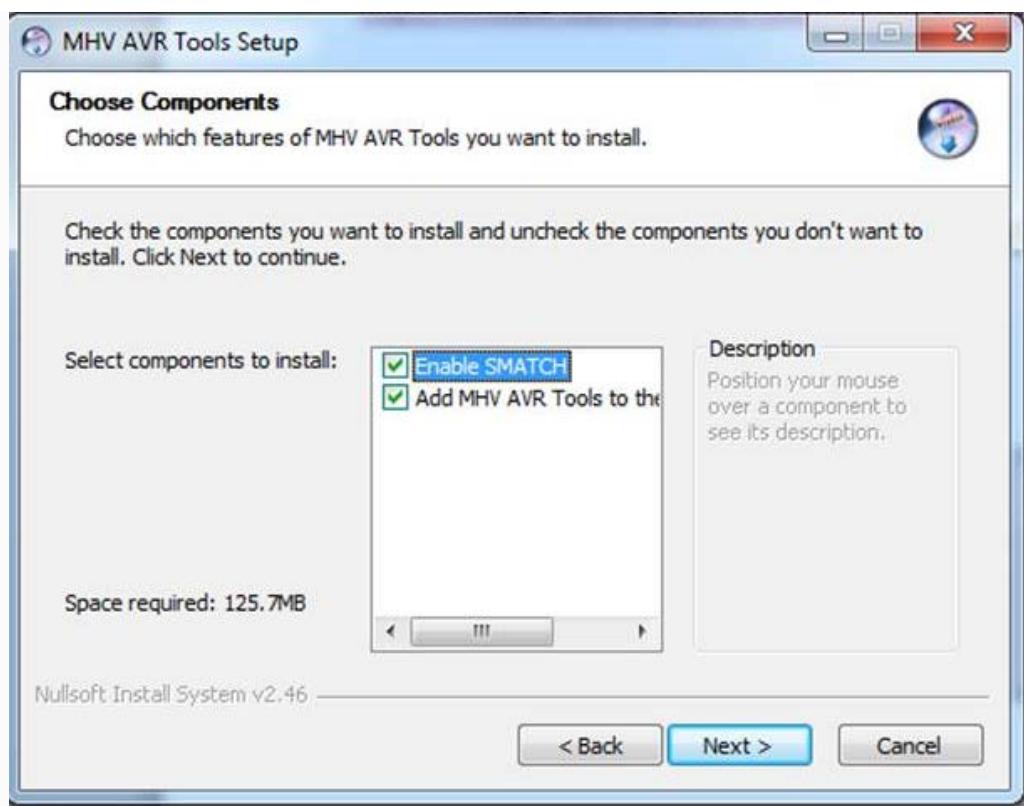
### Install MHV\_AVR\_Tools to its default location

1. Download and install the MHV\_AVR Tools:

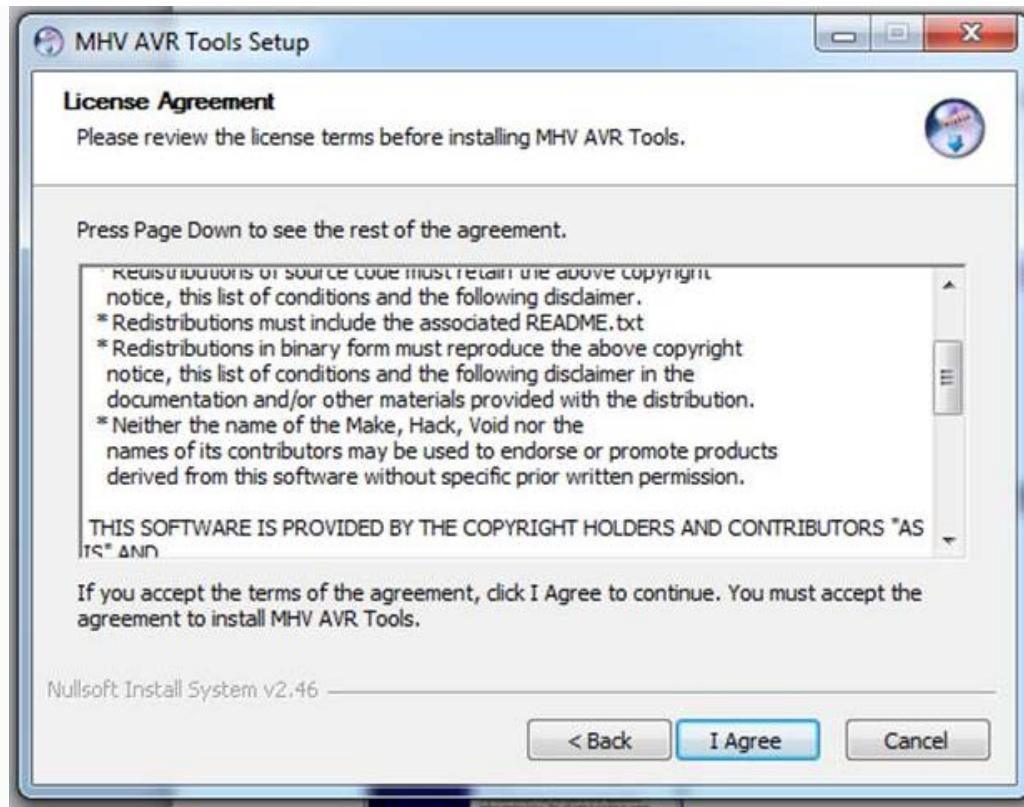
[http://firmware.ardupilot.org/Tools/Arduino/MHV\\_AVR\\_Tools\\_20121007.exe](http://firmware.ardupilot.org/Tools/Arduino/MHV_AVR_Tools_20121007.exe)



2. Select the **Next** button in the setup wizard screen then select the **Install** button for *MHV AVR Tools*



3. Check both items in the Choose Components Screen and select **Next** then select **Next** again to install to the default location



4. Select the **I Agree** button on the *License Agreement* screen.

### Install ArduPilot-Arduino

Download and unzip the ArduPilot Arduino package: <http://firmware.ardupilot.org/Tools/Arduino/ArduPilot-Arduino-1.0.3-gcc-4.8.2-windows.zip>

This can be unzipped directly to the **C:** drive or **C:\Program Files\**

#### Note

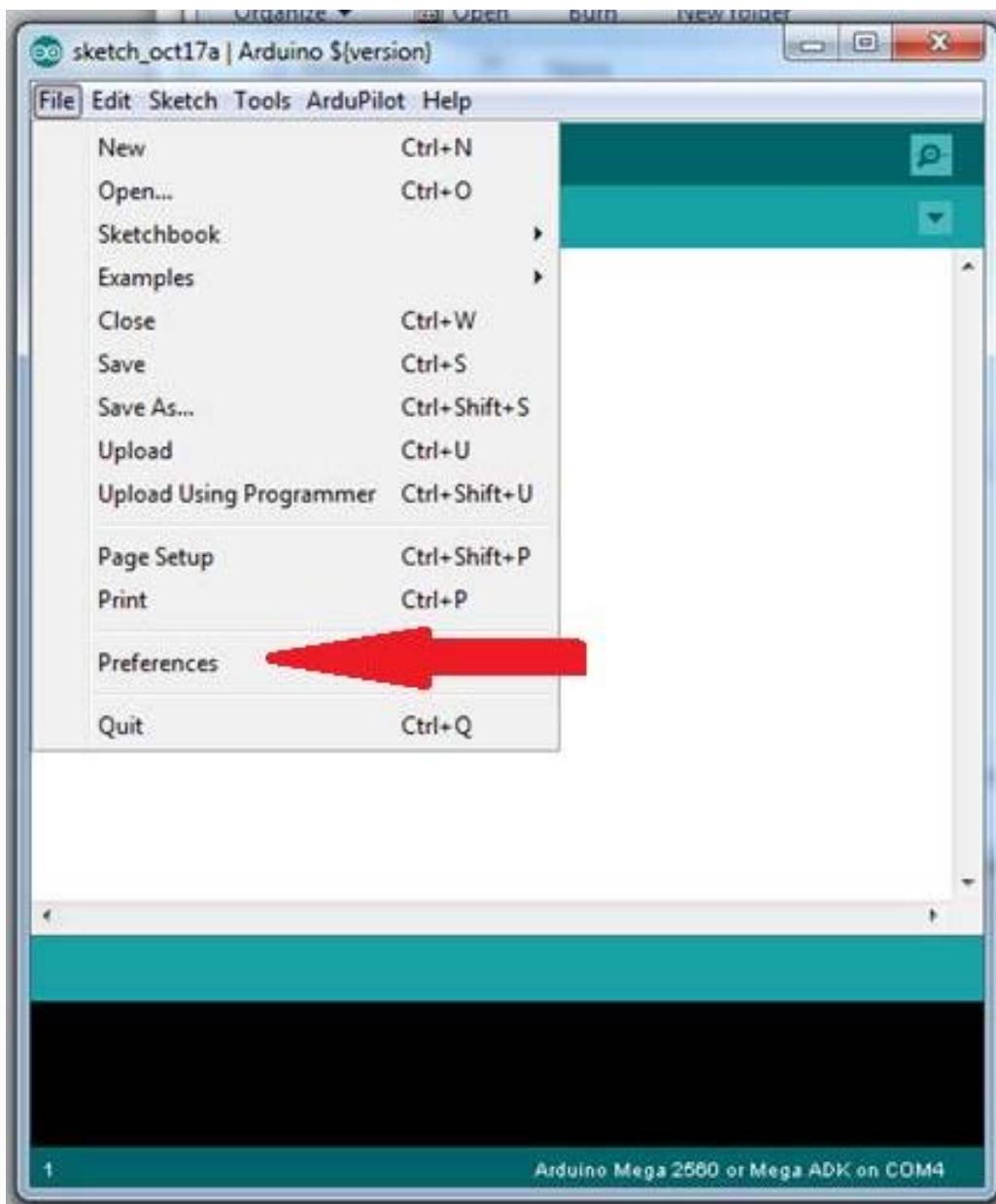
This is a special ArduPilot Arduino package which contains gcc 4.8.2

#### Configure Arduino

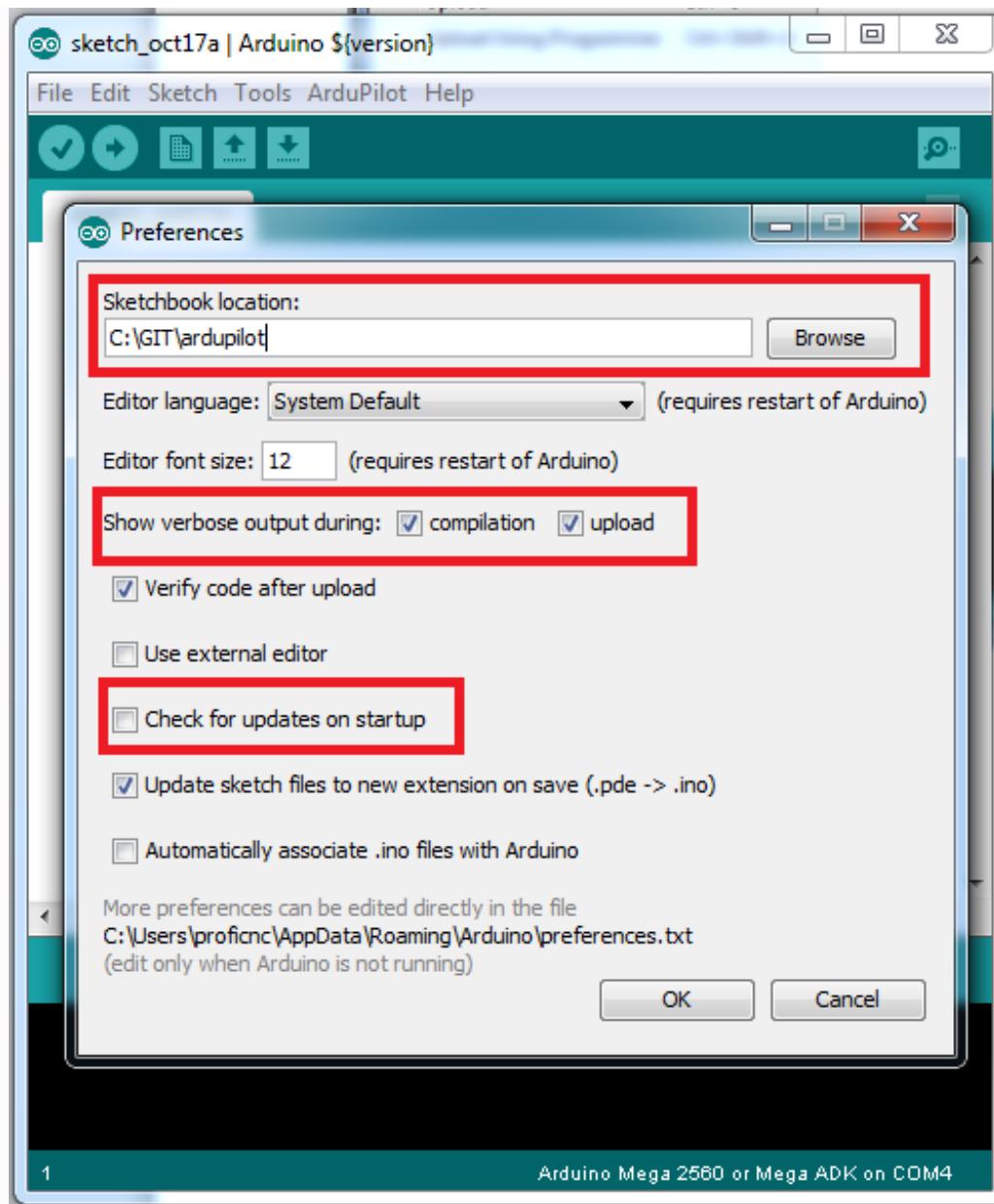
1. Go to your Arduino folder
2. Double click the Arduino icon



3. When Arduino opens, go to the file menu



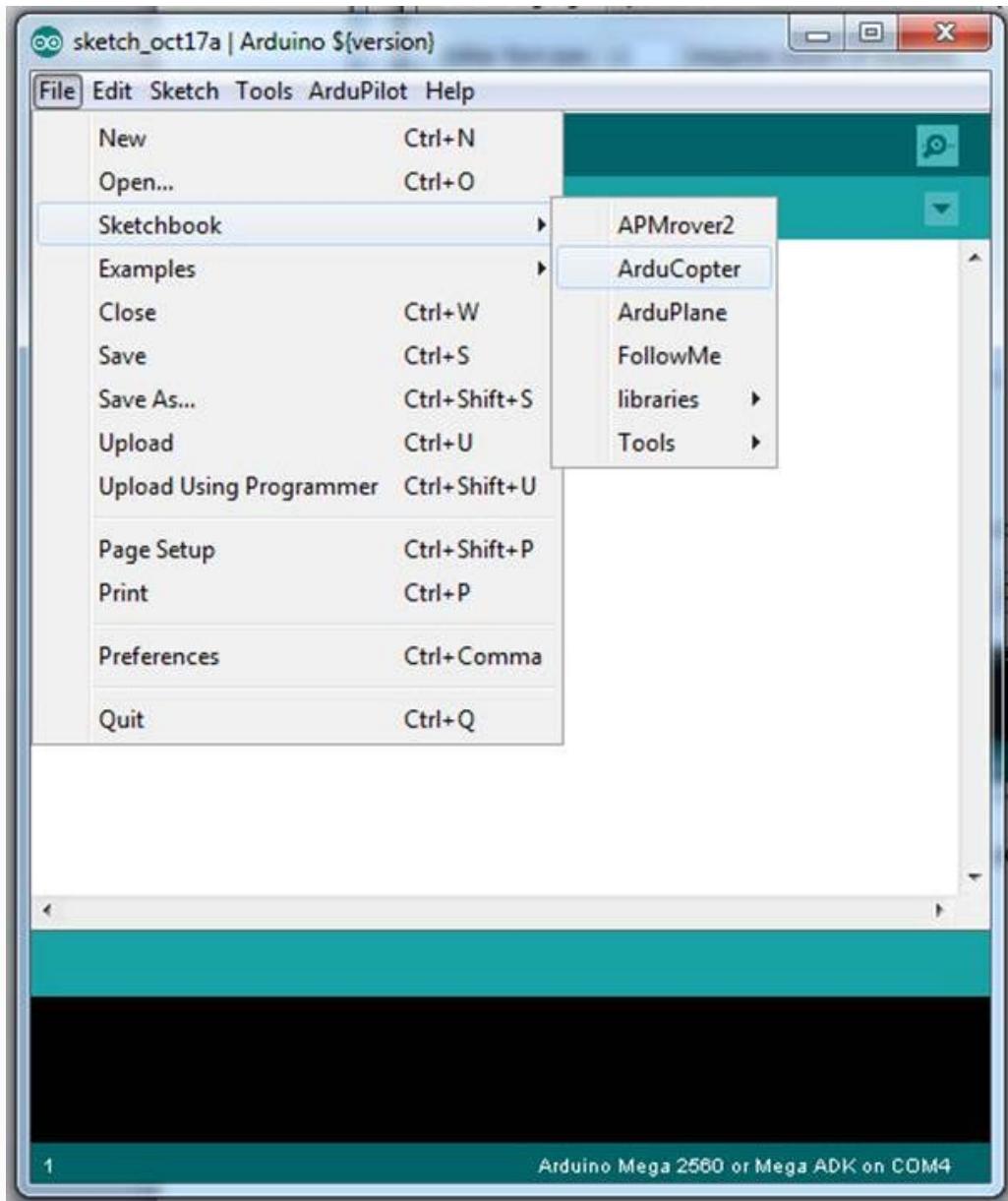
4. Select preferences



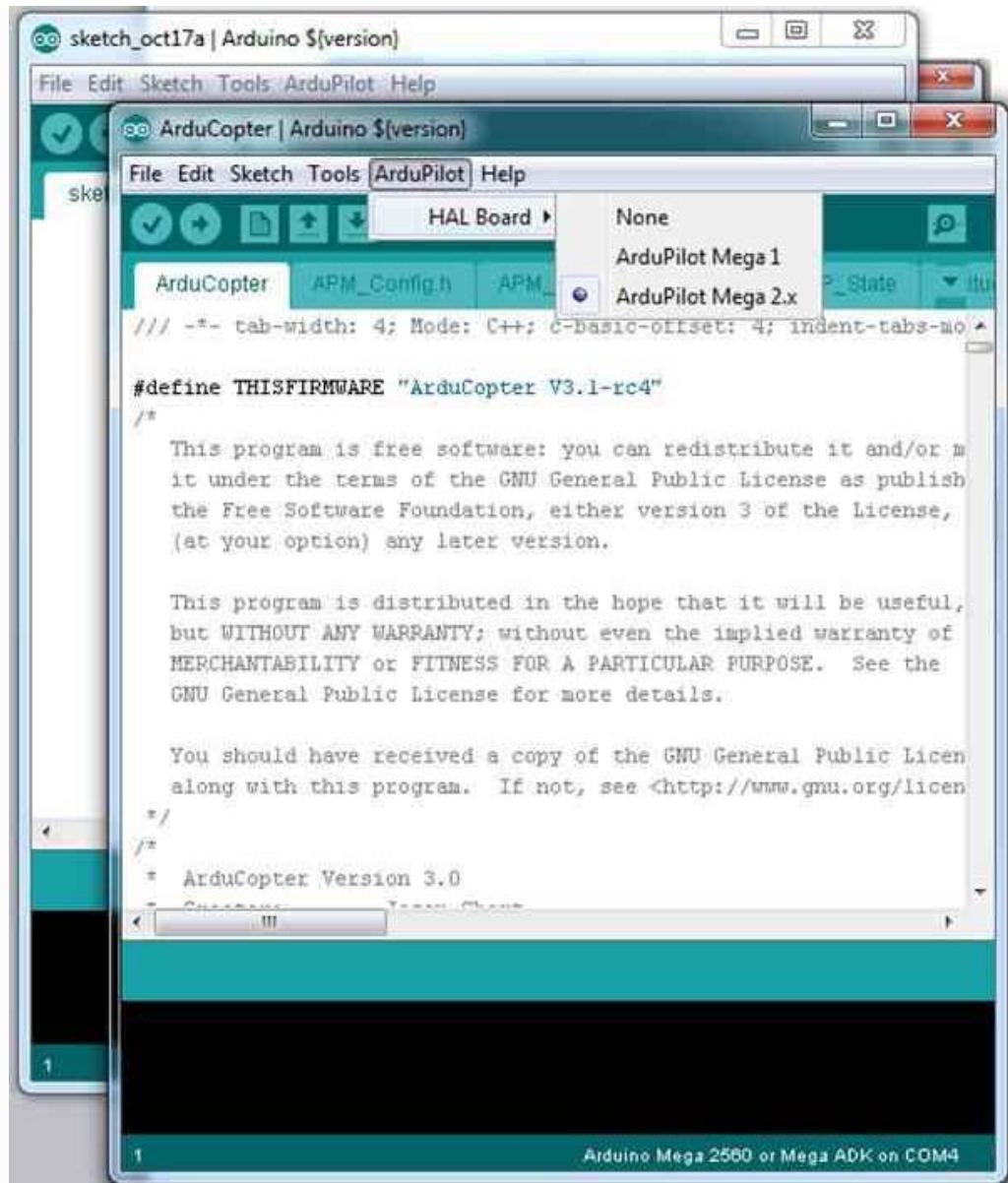
- Set Sketchbook location to your ArduPilot directory in your GIT folder.
  - Set verbose for both compile and upload
  - And DO NOT check for updates on start-up... (Remember, this is a special version just for ArduPilot.)
5. Click **OK** and close Arduino

### Connect your APM to your USB

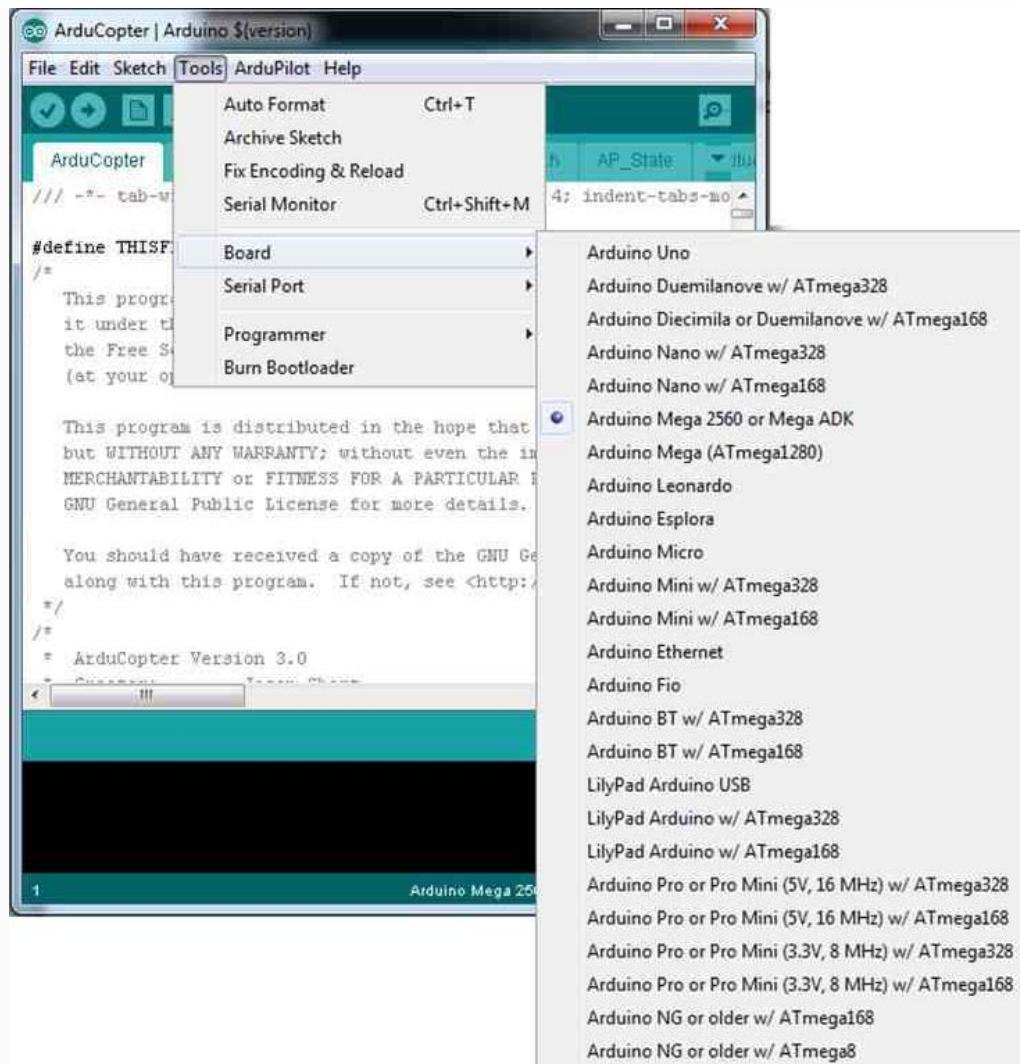
1. Re-open ArduPilot and under the file tab, click on sketchbook, then the program you wish to load onto your APM2.x (for this example we will use Copter, though the others use the same methods).



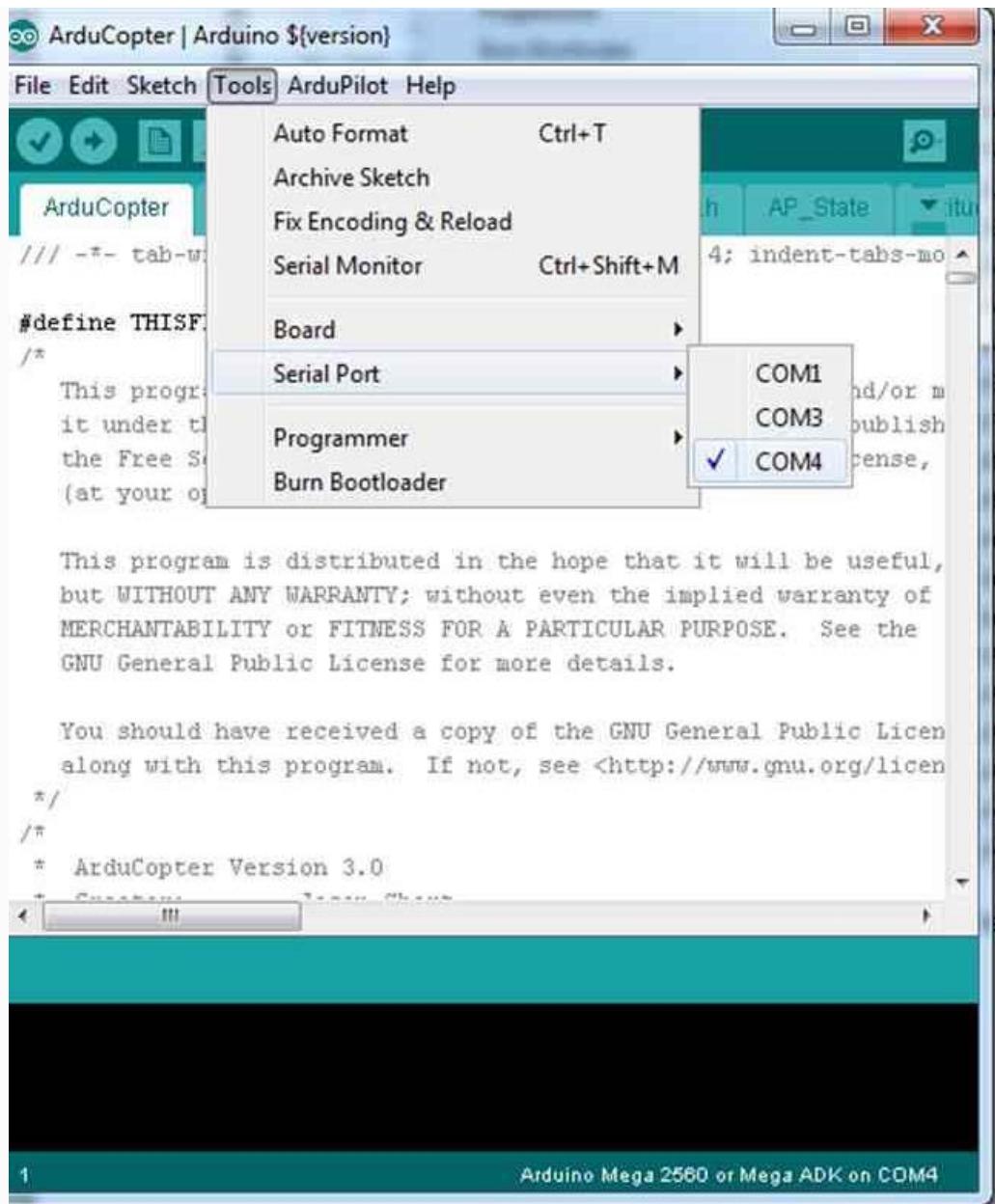
- Once this is loaded, click on the ArduPilot tab, and select ArduPilot mega 2.x out of the HAL options.



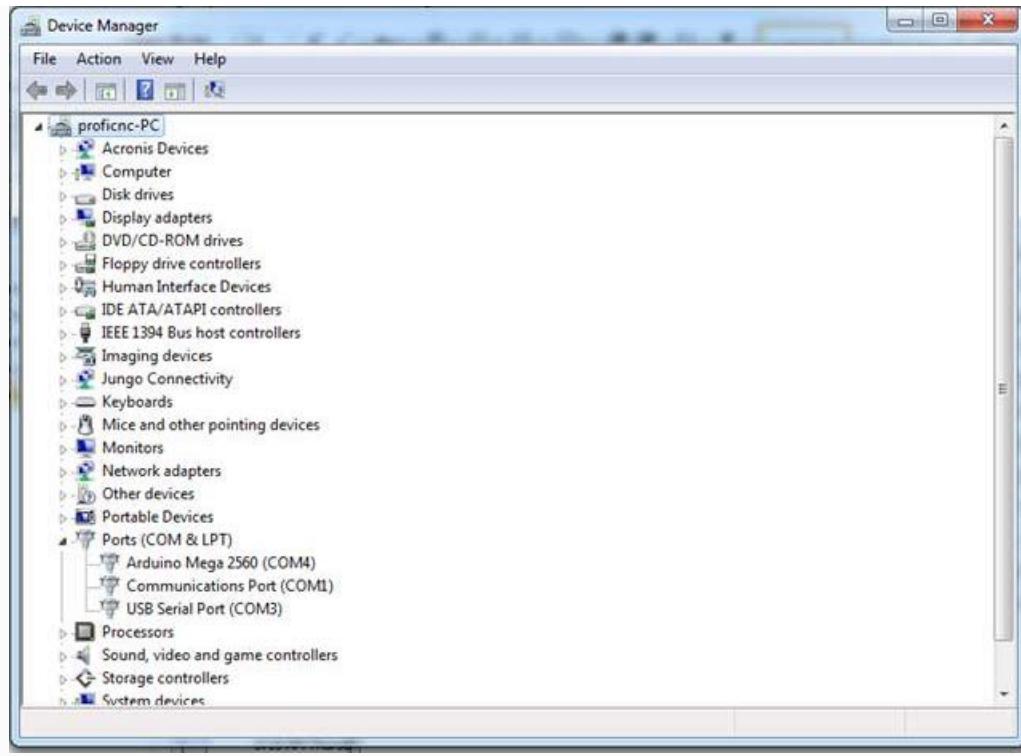
3. Then click the "Tools" tab and select "Arduino Mega 2560 or Mega ADK" from the "Board" tab.



4. Next select the *Tools* tab again, and set the "Serial Port" to the one your APM is connected to.



5. In my case it was COM4, but check under device manager / Ports to find out on your system.



## Configure Copter

1. Click on the **APM\_Config.h** file tab.
2. Set your frame type (e.g. `#define FRAME_CONFIG HEXA_FRAME`) in order to get the right image for your frame
3. Enable or disable the features you wish in this file.

If you want to compile with auto tune disabled, simply un-comment the line

```
//# AUTOTUNE DISABLED // disable the auto tune functionality to save 7k of flash
```

To disable Auto Tune which is enabled by default you would change it to:

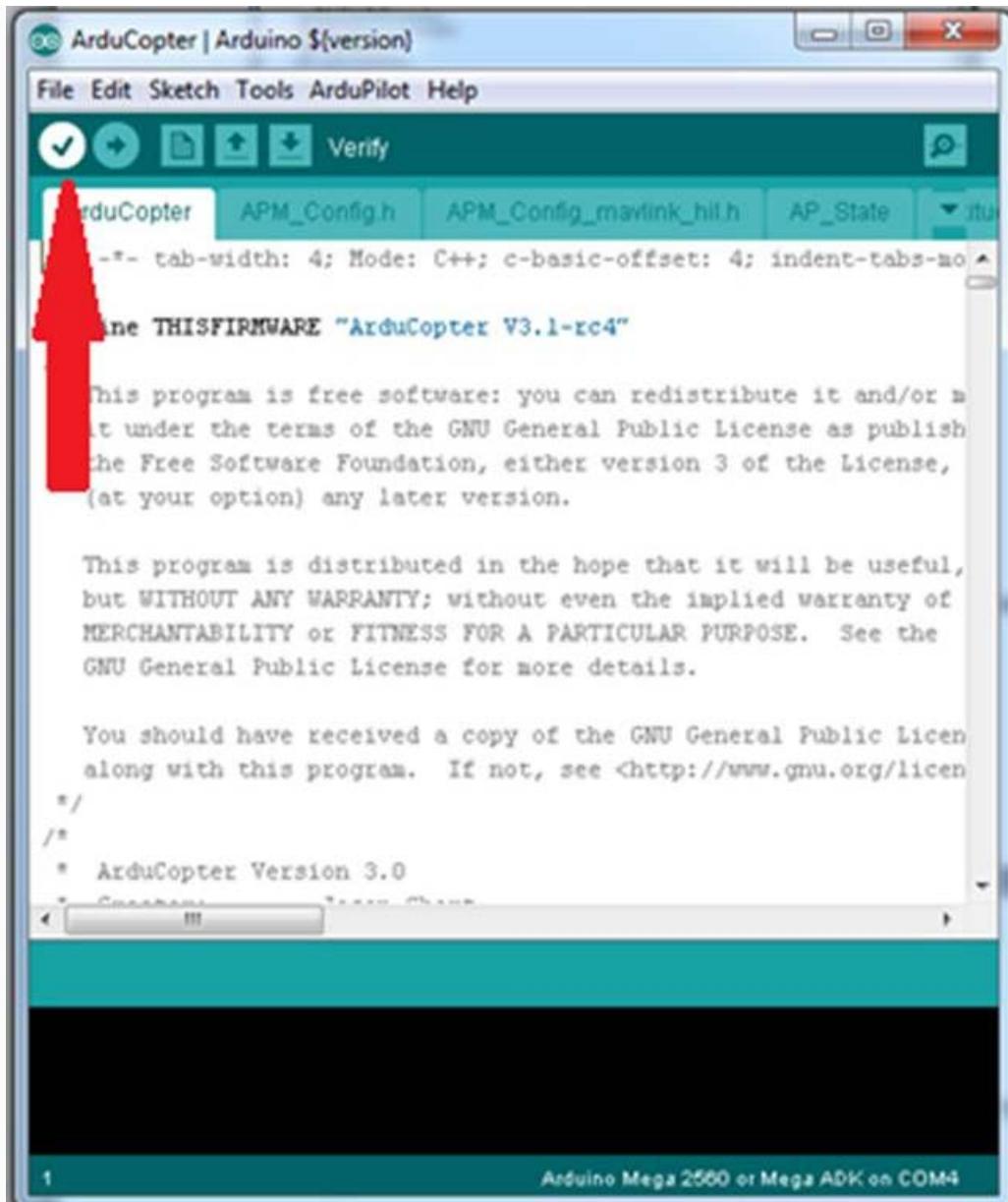
```
# AUTOTUNE DISABLED // disable the auto tune functionality to save 7k of flash
```

The commented out options are the NON-default and all that needs to be done is to un-comment them to use them instead.

4. Save this file and select the file Copter.

At this point you are ready to compile.

I would choose Verify for the first attempt.



ArduCopter | Arduino \${version}

File Edit Sketch Tools ArduPilot Help

Verify

ArduCopter APM\_Config.h APM\_Config\_mavlink\_hil.h AP\_State

```
-- tab-width: 4; Mode: C++; c-basic-offset: 4; indent-tabs-mode: t; c-file-style: "Linux";
```

```
define THISFIRMWARE "ArduCopter V3.1-rc4"

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License,
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

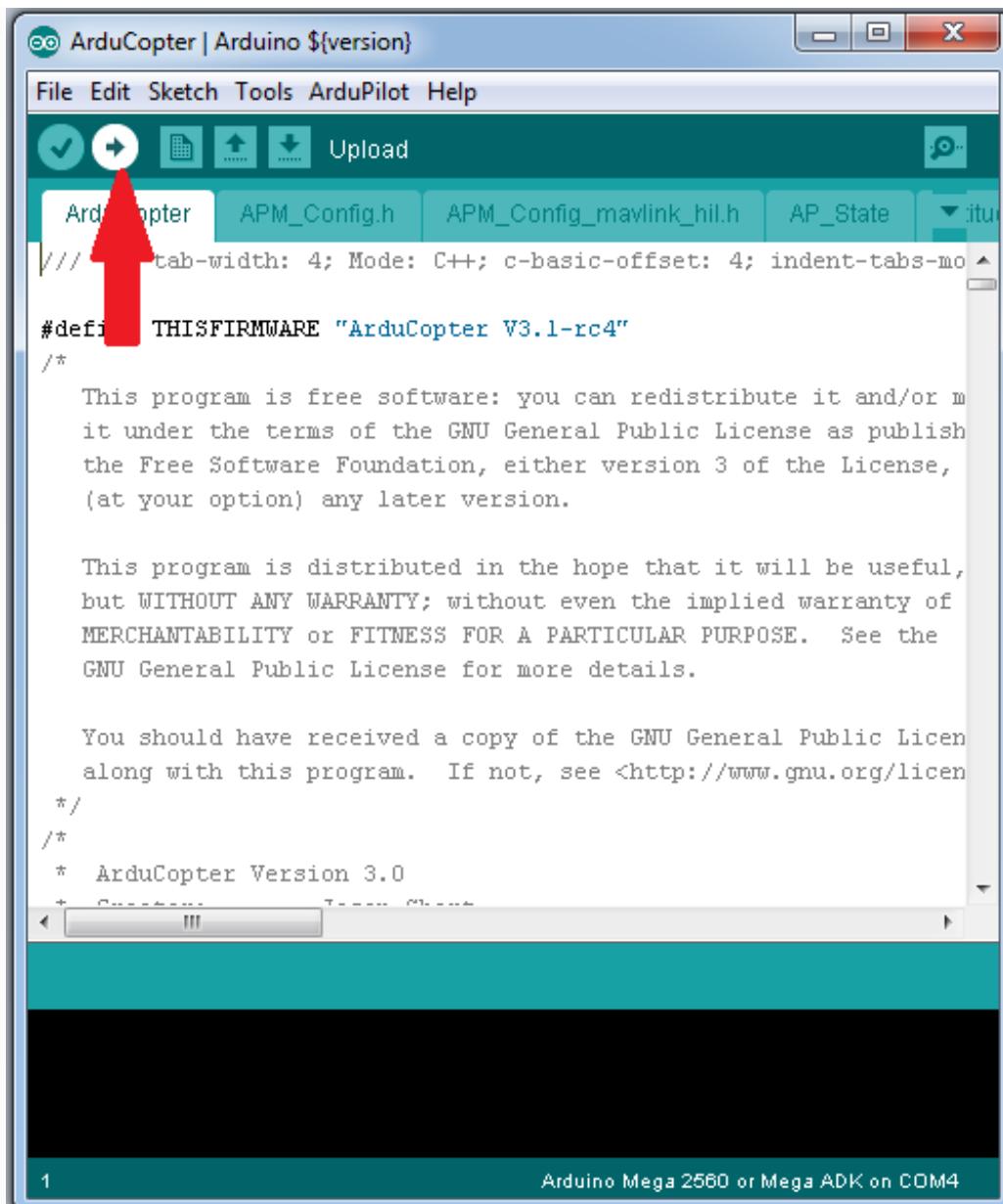
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
```

```
/*
 * ArduCopter Version 3.0
 */
```

1 Arduino Mega 2560 or Mega ADK on COM4

### Upload to your ArduPilot

1. Then if all is well upload to the autopilot, as shown:



This may take a while...

2. You should end up with the message as shown below.

The screenshot shows the Arduino IDE interface with the title bar "ArduCopter | Arduino \${version}". The menu bar includes File, Edit, Sketch, Tools, ArduPilot, and Help. The toolbar has icons for save, upload, and refresh. The tabs at the top show ArduCopter, APM\_Config.h, APM\_Config\_mavlink\_nll.h, AP\_State, and a dropdown menu. The code editor displays the ArduCopter sketch, which includes a license notice and a message about the program being free software under the GNU General Public License. The serial monitor at the bottom shows the output of the upload process: "Done uploading." followed by "avrduude done. Thank you." The status bar at the bottom right indicates "1" and "Arduino Mega 2560 or Mega ADK on COM4".

- Configure Your ArduPilot using planner, as normal.

#### Warning

The code you have just compiled is now UN-TESTED in your configuration. Please use only for testing. If you are not confident, please just use mission planner to upload pre-compiled code.

#### **Updating your code**

Please ensure that the version of code on your PC is the latest version, use git to update your code to the latest code.

#### **Building ArduPilot for APM2.x on MacOS with Arduino**

#### Warning

Copter 3.3 firmware (and later) no longer fits on APM boards. The last firmware builds that can be installed (v3.2.1) can be downloaded from here: [APM2.x](#) and [APM 1.0](#).

Plane, Rover and AntennaTracker builds can still be installed.

To build the ardupilot source code on MacOS for AVR targets (such as the APM1 or APM2) you have two choices. The first option is to build using a modified version of the Arduino build environment. You can get it from <http://firmware.ardupilot.org/> under the Tools directory.

The second choice is to build using the 'make' tool on the command line. If using the Arduino tool, then after installing it, you need to do the following:

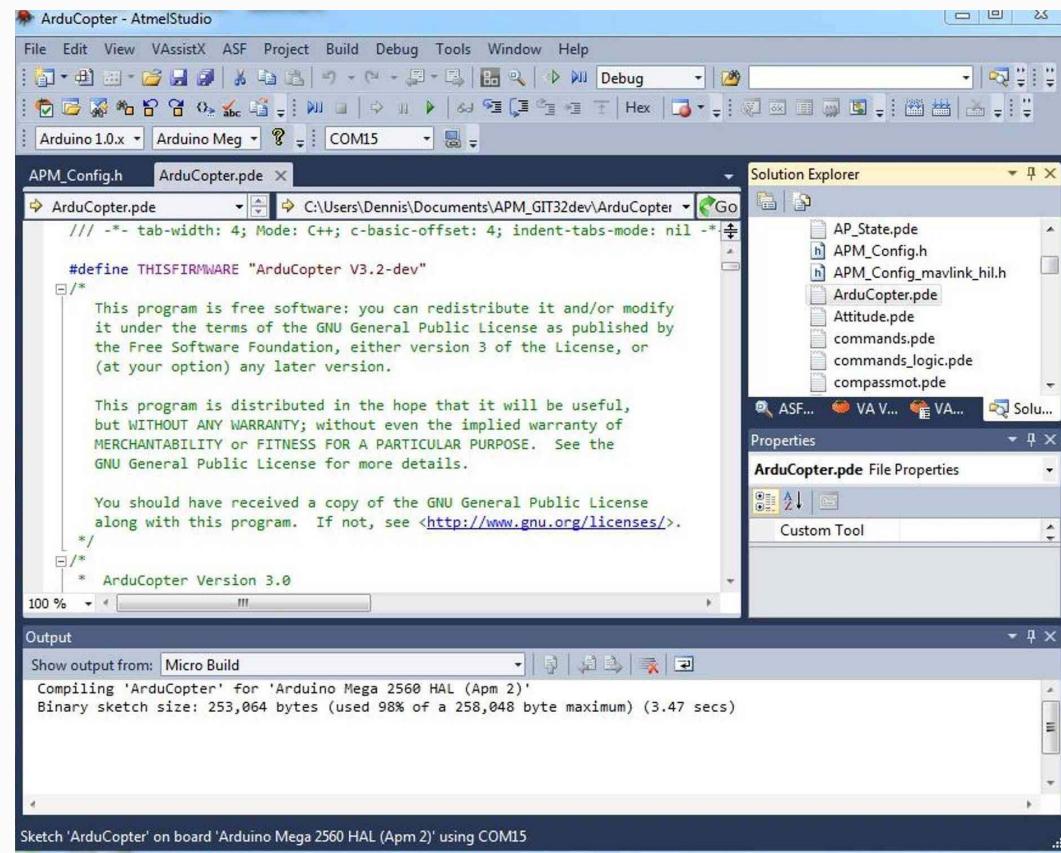
- Choose your board type under the ArduPilot menu
- Set your Sketchbook location under **File | Preferences** to point at the root of your git checkout of the ardupilot sources
- Stop and restart Arduino

## Building ArduPilot with Atmel Studio or Visual Studio & Visual Micro

This tutorial provides instructions for how to set up and build AMP using Atmel Studio 6.2 or Microsoft Visual Studio with the [Visual Micro](#) plugin. You can edit, build and upload the APM code using these tools. The builds and uploads are the same as with the ArduPilot Arduino builds.

### Introduction

Want a robust development environment for the APM code? Try out Atmel Studio 6.2 or Microsoft Visual Studio with the [Visual Micro](#) plugin. You can edit, build and upload the APM code using these tools. The builds and uploads are the same as with the ArduPilot Arduino builds.



Click the image to enlarge

**Overview:** Atmel Studio or Microsoft Visual Studio provides a robust Integrated Development Environment (IDE). However, the ArduPilot code is set up to use the special ArduPilot Arduino IDE and that structure does not work in those environments without some plugin to setup the correct structure and provide the up-load tools. Finding the source code for the various functions and classes is difficult using

Arduino - sometimes requiring a manual search through multiple folders and files to find the actual source code for one function or class used in the main code setup or loop.

**Solution:** Visual Micro solves most of these problems. Visual Micro is a plugin to Atmel Studio or Visual Studio that adds the following features:

- Compatibility with the Arduino Code structure, libraries, file naming and sketches - with the robust studio IDE.
- Uploading Arduino sketches to any of the Arduino boards - and, to the APM boards.
- All the files are shown in the solution panel, making it easy to open, browse or edit any file.
- Take a look at the [Visual Micro](#) website for all the details.
- Visual Micro has a debug feature when used with standard Arduino sketches but the APM code has removed or modified much of the standard Arduino features like Serial.print() and the hardware Serial features. The Visual Micro debugger does not work with APM. However, do not let that dissuade you from trying the Atmel Studio or Visual Studio IDE.

Here you will find a step by step process to set up Atmel Studio or Visual Studio to browse, edit, build and upload the APM code. Special thanks to Tim Leek at Visual Micro for developing this great plugin and for his help in creating this document.

## Setup

Even if you are not a current user of Atmel Studio or Visual Studio Pro, but have used Arduino IDE to build the APM code, then this process will get you started. Links to other wiki pages are provided where appropriate for more detail. No detail is included for Visual Studio or Atmel Studio features. You will need to learn that on your own.

The [Visual Micro](#) website explains how to install Visual Micro for Arduino and how to set up Atmel Studio or Visual Studio to work with Arduino sketches and boards. What is not clearly covered is how to add support for the APM boards. Once that information is clear, the set up is quite simple. Here is a summary of the key steps required:

- Assumptions:
  - The special ArduPilot Arduino must be installed and tested to build and upload the code. The complete details are [here](#). For this example it is assumed that the installation is at C:\ArduPilot-Arduino-1.0.3-windows.
  - You need [Atmel Studio 6.2](#) (free) or a licensed full version of Visual Studio Pro 2008, 2010 or 2012 installed. Visual Micro does not work with the free Express versions of Visual Studio. Visual Studio 2008 is used in this example.
  - You need the ArduPilot source code. For this tutorial the APM source code is assumed to be located in C:\Users\Public\Documents\ardupilot-Copter-3.1.3.
- Get the latest released code zip files “Plane x.x.x” [here](#) or use [Git Hub](#) to create a clone of the current code. If you want a specific release version go to the [diydrones/ardupilot](#) repository, select the desired ardulink branch, then click download zip.
- Download and install Visual Micro from the [Visual Micro](#) website and complete the [setup](#). Instructions are at the site.
- Test Visual Micro for Arduino using these [instructions](#). Get it working before you proceed.
- Download and install the APM board information. This is a key requirement to build and upload the APM code.
  - Go to the [Apm - Installation Guide](#) on the Visual Micro Forum. Read the post, then download the [boards.txt file](#).
  - Put the file “boards.txt” into a folder “APM” within the folder “Arduino/hardware”. Regardless of

where you Arduino IDE is located your Arduino folder (in this case C:\ArduPilot-Arduino-1.0.3-windows) should look like this:

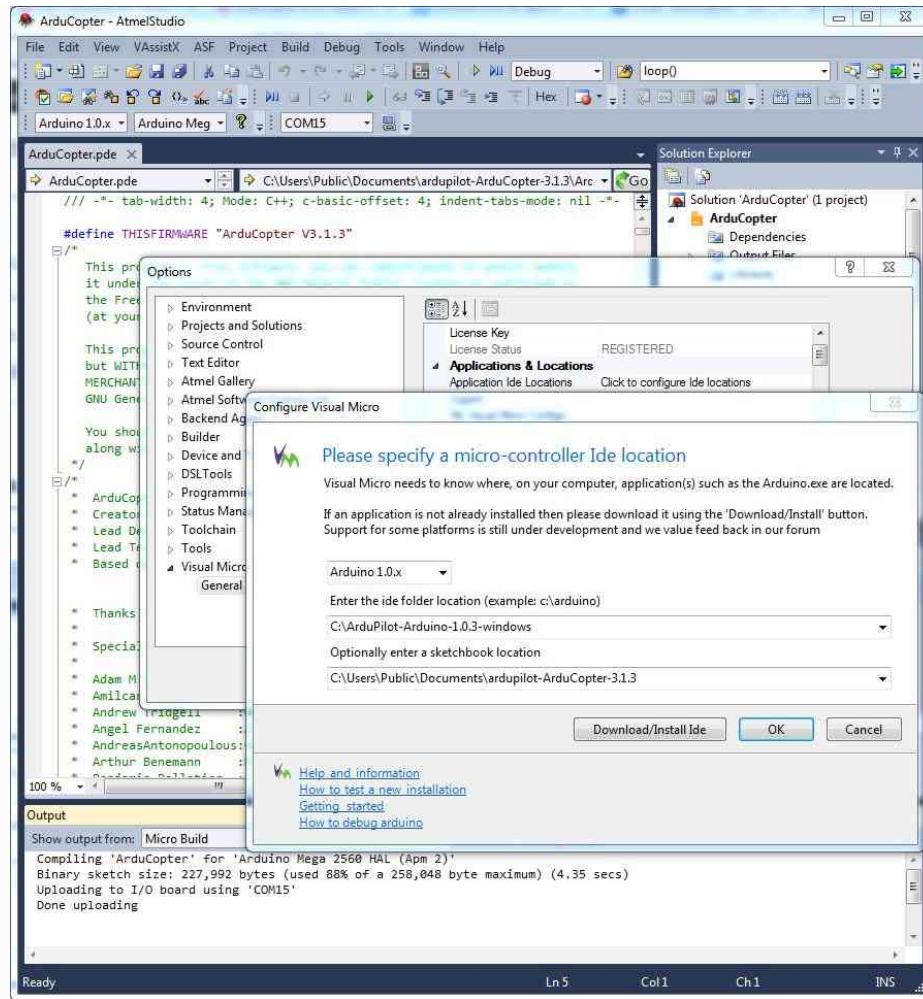


- Run Atmel Studio or Visual Studio and set up for APM as follows:

(APM2 is used in this example.)

- Tools>>Visual Micro>>Boards: Select Arduino Mega 2560 HAL (Apm 2)
- Tools>>Options>>Visual Micro: On right of window, select Applications & Locations>>Application Ide Locations>>Click to configure Ide locations: In the dialog window:
  - Select Arduino to “1.0.x: in the drop down text box.
  - Set the ide folder location to C:\ArduPilot-Arduino-1.0.3-windows (or the location of your Arduino IDE).
  - Set the sketchbook location to C:\Users\Public\Documents\ardupilot-Copter-3.1.3 (or the location of your ArduPilot code).
  - Note: This is the folder containing folders APMMover2, Copter, Plane, docs, FollowMe, libraries, mk and Tools.

Here is how the setup should look for this tutorial:



Open the arducopter.pde sketch: File>>Open>>Sketch Project:

Select Copter\arducopter.pde (or arducopter.pde file in you code folder.)

Build the code: Build>>Clean Solution then Build>>Build Solution. The code should build without error. Depending on what source code your are using, you may get the message "Sketch too big ...". You can reduce the size by un-commenting some of the compile options in APM\_Config.h.

**Important:** Each time you change a #define statement (or comment one or un-comment one) you must do Build>>Clean Solution followed by Build>>Rebuild Solution. Details are below in the Hints and Notes section.

### Uploading the code

After your code builds without errors you can upload the firmware to the APM.

- First do this:

Tools>>Options>>Visual Micro -Micro Debug - Advanced: Set Automatic debugging to False. Then F5 will upload without debugging - saving the need to press CTRL with F5.

- Connect your APM via the USB cable.
- Tools>>Visual Micro>>Serial Port. Set to the USB port detected for your APM. If the USB port is not detected, follow the Arduino installation instructions for adding the proper drivers. If the Arduino IDE works, then Visual Studio / Visual Micro should also work.
- To upload to the APM board, just press F5.

The build, and upload to the APM was flight tested by the author for the code release 3.1.3. The flight modes stabilize, altitude hold and loiter were all tested and behaved the same as upload of the same revision using Mission Planner. However, be aware that building your own upload from the source can result in unexpected results. You must configure all the options defines and other code correctly. Be careful, have fun and enjoy Visual Studio with Visual Micro.

## Hints and Notes

This procedure was tested to build and upload to the APM on the authors PC. There are other ways to configure Arduino, Atmel Studio, Visual Studio, and Visual Micro. Those methods are left to the reader. For instance, you can install the ArduPilot Arduino IDE in any location, not just in the Programs area. It is suggested to use this process first to verify everything works.

**Ways to reduce the size of Copter so it will build:** In Visual Studio or Atmel Studio solution panel (where the source files are listed), open the file Copter/Header Files/APM\_Config.h and un-comment some of the #Define XXX DISABLED lines to save some space. I.E. disable MOUNT, OPTFLOW, CAMERA,CONFIG SONAR and/or PARACHUTE as appropriate for your APM. Each time you change a define you must do Build>>Clean Solution followed by Build>>Rebuild Solution.

**Errors building from fresh source clone:** The first time you set the application location (for instance after getting a fresh clone of the source) or change the IDE you are using, you may get build errors even though you have selected the correct Arduino board. To get an error free build, just re-select the correct board. See Multiple installs of Arduino below for one reason this may occur.

**Fast compile vs changing defines:** Visual Micro has a default option for fast compiles. This is setup as the default. The IDE checks for changes to files and if there are no changes, does not recompile unchanged files. This really great feature has a side effect when define statements are changed because they effect other files but do not specifically change the text (code) thus files effected by defines will not be recompiled causing a real mess. There are two ways to avoid this issue:

- Each and every time you change a define or un-comment a define or comment a define do this:
  - Build>>Clean>>Solution (That will clean out the pre-compiled cache)
  - Build>>Rebuild
  - All following builds (assuming no changes to defines) can be the really fast version Build>>Build Solution
- Or, if you like to wait a long time for each compile you can change the Visual Micro options
  - Tools>>Options. Select Visual Micro. scroll down to Compiler Optimisation. Hey, in UK they spell it that way.
  - Set Core Modified and Library Modified to False

**Multiple installs of Arduino:** It is important to note that Arduino has only one location where the parameters are stored - in C:\Users\...\AppData\Roaming\Arduino\preferences.txt. Any time you start any installation of Arduino - a standard version, the Arudpilot Arduino version or even changing settings in Visual Micro - that file may get changed. So, it is very important to check all the settings in the IDE you are using each time you change the IDE - to avoid having the preferences set to that of the last IDE you used.

**Referencing a standard Arduino in Visual Micro:** Normally, Visual Micro is setup to reference a standard installation of Arduino instead of the special Arudpilot Arduino for the HAL versions of APM. You can configure Visual Studio or Atmel Studio to reference a standard installation and it may build and upload the APM code but that upload will not work. The build size is different and it does not connect to Mission Planner. It is strongly suggested you only reference the special Arudpilot Arduino installation when working with APM code.

**Using Arduino statements and libraries:** This is better stated as not using standard Arduino statements. The newer HAL versions of ArduPilot for the APM boards removed most, if not all, of the

standard Arduino statements and libraries. You can not just add an analogRead(sensorPin) statement for instance. So, do not be frustrated if you try to use the standard Arduino language reference to edit the APM code - it will mostly not work. There are equivalent calls and statements for the APM but you will have to search for examples in the code.

**Building Older Versions of ArduPilot:** If you are still working with version 2.9.1b or possibly older versions prior to HAL, you can take advantage of the Atmel Studio or Visual Studio and Visual Micro IDE.

You just need to do the following:

- Download the version of [ArduPilot Arduino that supports the 2.x.x revisions](#). Don't confuse this with the current version. They have the same name but are different.
- Install the ArduPilot Arduino in a separate folder, and add the `apm/boards.txt` file into the hardware folder - like the above instructions.
- In Atmel (or Visual) Studio, in Tools>>Options>>Visual Micro, set the Applications and Arduino locations to reference the ArduPilot Arduino that supports the 2.x.x revisions, and to the folder containing the 2.x.x source code. You can get the 2.9.1b code at the [diydrones/ardupilot](#) repository, select the desired ardupilot branch, then click download zip.
- Set Tools>>Visual Micro>>Boards to APM Arduino Mega 2560.

## Building for NAVIO+ on RPi2

### Overview

These instructions clarify how to build ArduPilot for the NAVIO+ board on the NAVIO+'s RPi2 board itself using Waf build system. These instructions assume the RPi2 has already been setup according to the manufacturer's (i.e. Emlid's) instructions [here](#).

Alternatively you can follow Emlid's instructions on how to build from source found [here](#).

#### Setup

Use an ssh terminal program such as [Putty](#) to log into the NAVIO+ board's RPi2.

Clone the source:

```
git clone https://github.com/diydrones/ardupilot.git
cd ardupilot
git submodule update --init
```

#### Note

Waf should always be called from the ardupilot's root directory.

To keep access to Waf convenient, use the following alias from the root ardupilot directory:

```
alias waf="$PWD/modules/waf/waf-light"
```

Choose the board to be used:

```
waf configure --board=navio
```

#### Build

Now you can build arducopter. For quadcopter use the following command:

```
waf --targets bin/arducopter-quad
```

To build for other frame types replace quad with one of the following options:

```
coax heli hexa octa octa-quad single tri y6
```

In the end of compilation binary file with the name arducopter-quad will be placed in  
[ardupilot/build/navio/bin/ directory](#).

## Building for NAVIO2 on RPi3

### Overview

These instructions clarify how to build ArduPilot for the Navio2 board on the Navio2's RPi3 board itself using Waf build system. These instructions assume the RPi3 has already been setup according to the manufacturer's (i.e. Emlid's) instructions [here](#).

Alternatively you can follow Emlid's instructions on how to build from source found [here](#).

#### Setup

Use an ssh terminal program such as [Putty](#) to log into the Navio2 board's RPi3.

Clone the source:

```
git clone https://github.com/diydrones/ardupilot.git
cd ardupilot
git submodule update --init
```

#### Note

Waf should always be called from the ardupilot's root directory.

To keep access to Waf convenient, use the following alias from the root ardupilot directory:

```
alias waf="$PWD/modules/waf/waf-light"
```

Choose the board to be used:

```
waf configure --board=navio2
```

#### Build

Now you can build arducopter. For quadcopter use the following command:

```
waf --targets bin/arducopter-quad
```

To build for other frame types replace quad with one of the following options:

```
coax heli hexa octa octa-quad single tri y6
```

In the end of compilation binary file with the name arducopter-quad will be placed in  
[ardupilot/build/navio2/bin/ directory](#).

## Building Mission Planner with Visual Studio

### Introduction

Mission Planner (MP) is an open source ground station developed in C# primarily for use on Windows computers (although it can be run on Mac using mono). This is the most commonly used ground station as it provides the most complete functionality for vehicle setup as well as pre-flight mission planner, in-flight monitoring and post flight log file analysis.

This page provides instructions on how you can build the Mission Planner software on your own machine using MS Visual Studio 2015 which may be useful if you wish to make changes for your own use or improvements for the community. Building the mission planner may also help as a guide in case you plan to build your own custom ground station development.

Some warnings before you dive in:

- Use your modified / compiled version of Mission Planner at your own risk.
- Mission Planner is a very complex including and making changes is not for the faint of heart. Here are the basic skills you will need to make reasonable progress with MP changes:
  - C# programming skills and experience (at least or C++ experience).
  - Experience with Microsoft Visual Studio (VS) development environment. MP is not the application to begin learning VS.
  - Experience using Windows API (Application programming Interface) - including understanding of streams, processes, threads.
- Support for Visual Studio, programming in C# and Windows API may not be forthcoming from the DIY Drones community. You will need to get that support from other sources.

### System Requirements

Here is what you will need.

- Windows XP, Vista, 7, 8, 10
- Sufficient disk space, memory, processor power to comfortably run Visual Studio (details below)
- An Internet connection.
- Visual Studio 2015 community edition

### Install Visual Studio and DirectX and Python

The first step is to get [Microsoft Visual Studio 2015](#) installed and working in your Windows system.

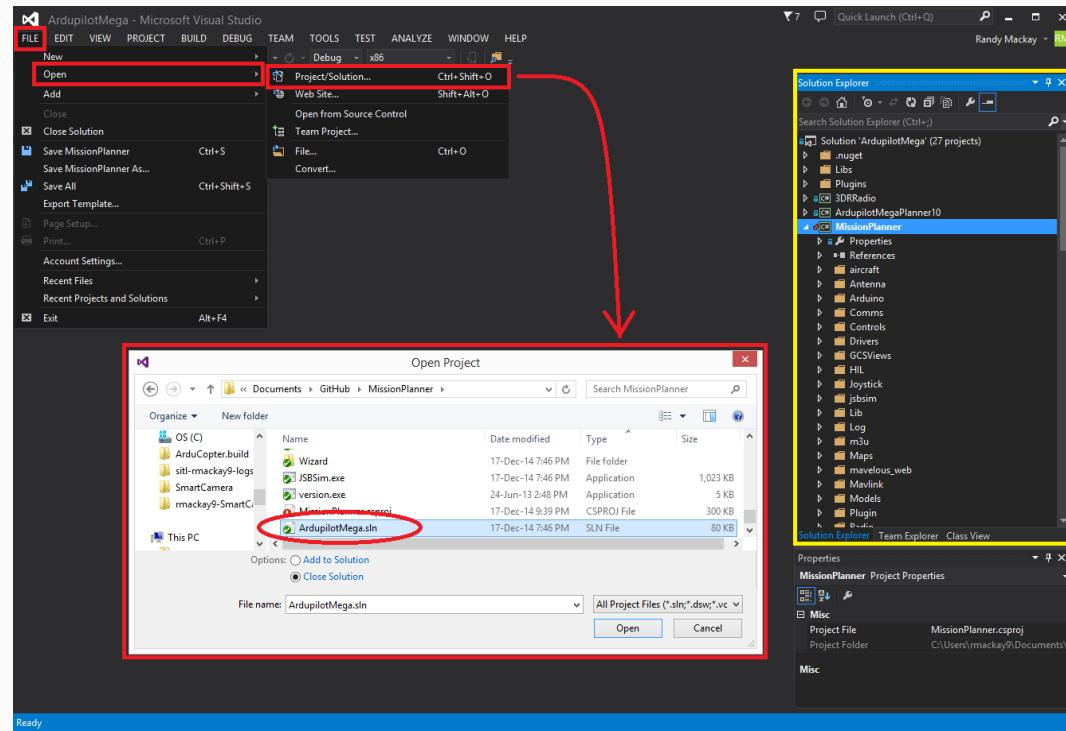
- Download and install MS Visual Studio 2015 Community Edition which can be found [here](#).
- Reboot your PC
- Start Visual Studio from the Start Menu
- After your installation is complete and before attempting to work with Mission Planner test your installation on a simple "Hello World" application.

Install Python 2.7 (if not already installed) by downloading it [here](#).

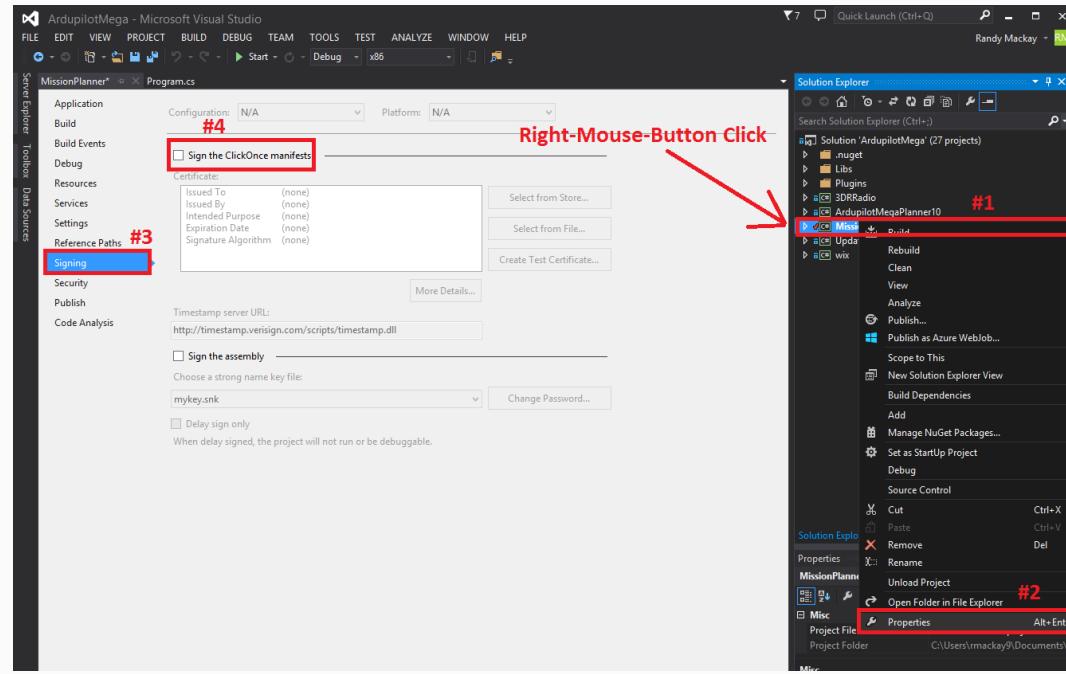
### Getting the source code from Github into your computer

The Mission Planner source code is stored in GitHub. In general you can follow the instructions [for the ardupilot flight code](#) except that you should use the <https://github.com/ArduPilot/MissionPlanner> repository in place of the ardupilot repository.

## Open the Mission Planner solution in Visual Studio



- Start Visual Studio
- Click File >> Open >> Project / Solution
- Navigate to where the Mission Planner source was downloaded to and open MissionPlanner.sln.
- Visual Studio should open the “solution” which includes the Mission Planner and a few other related applications (i.e. “3DR Radio”, “Updater”, etc) which can all be see in the Solution Explorer (highlighted in yellow above).
- Set the “Solution Configuration” to “Debug” or “Release” (this can be found just below the Tools menu)
- Set the “Solution Platforms” to “x86”
- In the Solutions Explorer, right-mouse-button click on Mission Planner and select Properties, Signing and uncheck “Sign the ClickOnce manifests”



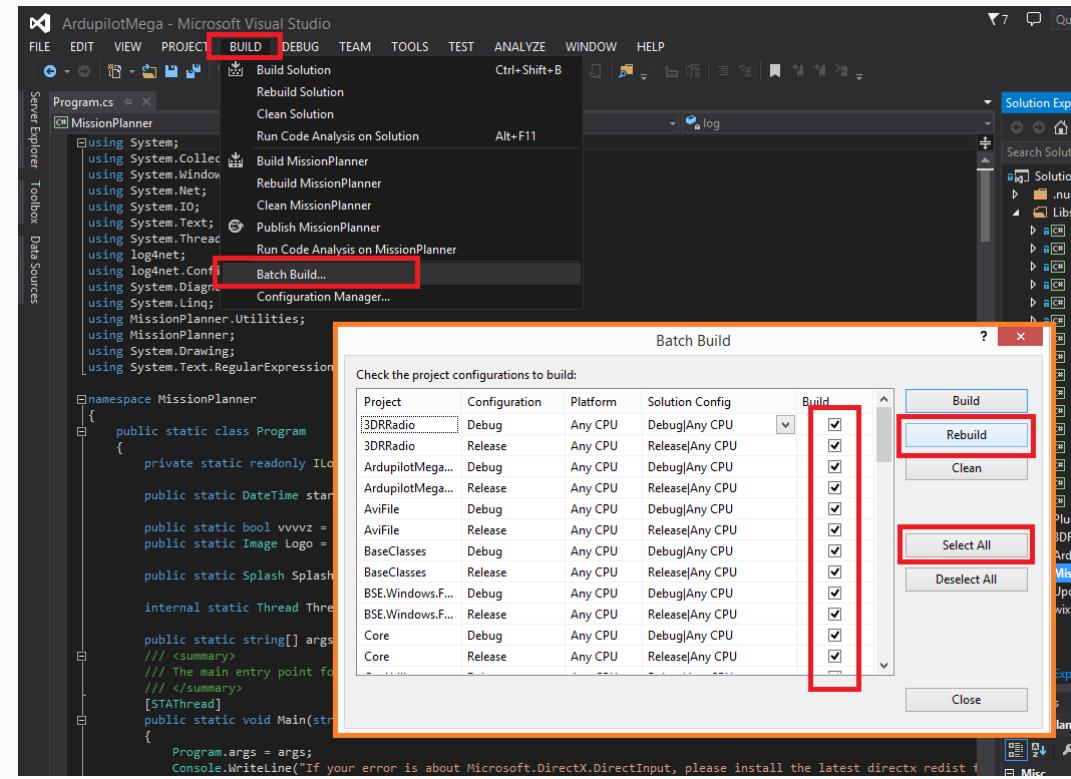
- Mission Planner is made up of several projects, you can see these by expanding the “MissionPlanner”

and "Libs" folders of the Solution Explorer.

- MissionPlanner (the main code)
- AviFile
- BaseClasses
- BSE.Windows.Forms
- Core
- GeoUtility
- GMap.Net.Core
- GMap.Net.WindowsForms
- KMILib
- MAVLink
- MetaDataExtractor
- MissionPlanner.Comms
- MissionPlanner.Controls
- MissionPlanner.Utils
- px4uploader
- SharpKml
- ZedGraph

## Building Mission Planner

Before you attempt to build (compile) Mission Planner you must also have the official version installed on your PC. This is because there are some .dll files that are not included in the Git repository.



- Select Build >> Batch Build..., "Select All" (to check all checkboxes) and then press "Rebuild". You will probably see errors on your first attempt to compile (build) Mission Planner so try a couple more times.

If errors persist try some of the following:

- For errors related to missing dlls:
  - In the Solution Explorer right click the MissionPlanner project, Properties, Reference Paths
  - In the Folder entry, browse to and select the location of the "installed" Mission Planner which is

probably:

C:\Program Files (x86)\Mission Planner OR C:\Program Files\Mission Planner

- Click the Add Folder button to put the path to the installed MP into the Reference paths box.
- Click (select) Build Events. Remove all pre-build and post build options.
- Click (select) Build.

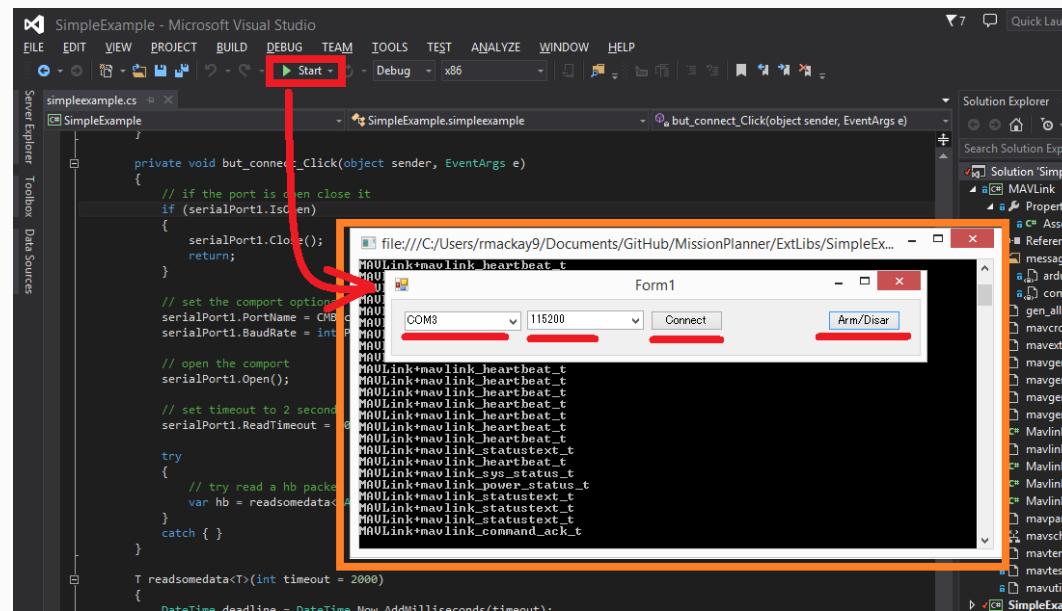
For errors about missing references, you will see the name of the project for each error listed. Select Properties for each project with such errors and add the location of the Installed Mission Planner like you did above for MissionPlanner project. That should reduce the errors.

If you see an error in project BSE.Windows.Forms "...could not locate the Code Analysis tool at ". You can eliminate this by un-checking the Enable Code Analysis box in Code Analysis in the BSE.Windows.Forms properties.

Some optional help in resolving build errors:

- In VS, Select menu items [BUILD] [Configuration Manager] This will show you which projects are compiled (built) each time you do a build or re-build solution.
- Check 'Build' for any that are not checked:  
(I.E. 3DRRadio, Updater, wix)
- Do [Build], [Clean Solution] then [Build], [Rebuild solution].
- All projects should build without errors.
- When you build without errors, you are ready to begin browsing or editing.

## Building the SimpleExample



The "SimpleExample" solution is available as a near minimal application to demonstrate how a C# program can connect to a vehicle and cause it to arm or disarm. This example has many fewer dependencies than the full Mission Planner and is simpler to build and understand.

Open the solution from Visual Studio by selecting File >> Open >> Project/Solution, and in the MissionPlanner code directory select ExtLibs / SimpleExample.sln

Ensure the program can be build successfully by selecting Build >> Build Solution.

After first checking that you can connect to your flight controller and arm it with the regular mission planner, disconnect the regular Mission Planner and then press "Start" to run the application in debug mode. When the "Form1" pops up, select the COM port, the baud rate (probably 115200) and press Connect. If it successfully connects, press Arm/Disarm to attempt to arm the vehicle.

Note: there is no error checking in the application so if it fails to connect it

### **Editing and Debugging Mission Planner (and Other Tips)**

Editing and debugging details are beyond the scope of this Wiki. Debugging may result in some warnings. You should learn what they mean and take the necessary steps to resolve them if that is the case. Here is a simple debugging example to get you started.

- Do not connect your APM to the compiled version of MP. You must first copy some .xml files to the bin/debug folder. See details below.
- First be sure VS is configured for debug (versus release) Set this in the top menu tools area or the configuration manager.
- Select menu DEBUG, Start Debugging. (Or, press F5). Mission Planner should run as you normally see it. However, some important configuration files are missing so connection to the APM is not recommended at this time.
  - If after "Start Debugging" the program loading hangs in the splash screen and you see this message: "Managed Debugging Assistant 'LoaderLock' has detected a problem ..... " and/or the debugger has paused at the line Application.Run(new MainV2()); in ArduPilotMega.Program then do this:

Select [Debug], [Exceptions]. Expand the [Managed Debugging Assistants]. Uncheck the 'Loader Lock' check box

- Close MP. (Or, select menu DEBUG, Stop Debugging in VS).
- Next you can try setting a break point.
  - Expand the MissionPlanner project in the VS Solution Explorer so you see the objects included.
  - Scroll down to MainV2.cs, right click that object and select View Code.
  - In the code window for MainV2.cs, scroll down to the module public MainV2() then to the line
  - splash.Text = "Mission Planner " + Application.ProductVersion + " " + strVersion; (about line number 169)
  - Click in front of that line (In the dark gray bar on the left) to set a break point (red circle).
  - Start Debugging (press F5).
  - You will see the normal MP start up windows up to the Splash window but then it will stop running. You have hit the break point. Visual Studio will show the code and the break point will be highlighted. Note that you cannot move the splash screen so you may need to relocate the VS window to see the break point.
  - Move your mouse over different variables and objects in the code. You will see the current values of many or the items.
  - Press F5 and Mission Planner will continue loading.
- Further details on editing and debugging are left to the user.

### **Using your modified Mission Planner**

If you make changes to Mission Planner, you will probably want to make use of your version. Here we will give you some preliminary information to do that. You can use your local compiled version but the compiled output files are located in different places in VS and some additional steps are required. There are configuration files specific to your installation of Mission Planner that are not included in the Git hub download that are only provided in the Mission Planner installation package. You will need to copy these

to the correct area in the folder you are using for the Visual Studio project. Here are the steps that will get you started.

- **Use your modified complied version of Mission Planner at your own risk.**
- **These steps assume VS is in the debug configuration. [editors]**  
Details when in Release mode could be added [/editors]
- In order for your VS version of MP to function with the APM connected, you will need to copy several files from the folder where MP is installed (C:\Program Files (x86)\APM Planner or C:\Program Files\APM Planner) to the folder where your VS project compiled output is located.

Todo

editors: This needs to be made more accurate which files are needed, why etc.

- **Copy (don't move) all xml files** (I.E. files with the extension .xml) from the root folder of the MP installation (C:\Program Files\APM Planner) **to the bin/Debug folder** in the folder where your Visual Studio Mission Planner solution is stored. (the Git hub clone folder). This will setup your compiled version to match the current configuration of your APM (copter versus plane, other options.)
- I.E, if your solution is in folder MPGITClone, then copy the .xml files to MPGITClone\bin\Debug.  
Some will copy without notice, but some will ask you if you want to replace the existing file.  
Replacing all seems to work but you should investigate further to be sure you can use MP for real life situations before you do so.
- If you build Mission Planner in Release mode, then the files should be copied to the bin/Release folder. This has not been tested at this time.
- Here are some other tips:
  - Location of Logs saved when using your version will be in the /bin/Debug or bin/Release folder.  
This can be changed with Mission Planner 1.2.63 and later versions.
  - If you want to make a shortcut to run your version of Mission Planner without running Visual Studio, create the shortcut to point to the program ArduPilotMegaPlanner10.exe in the bin/Debug or bin/Release sub folders.
- At this point your local version of MP should be working. You should be able to connect to your APM, Flight Data including status should work, Configuration should bring up your APM parameters, Terminal should work including saving and browsing logs. Flight Planner should also work. As mentioned before, use your modified version at your own risk.

## Submitting your changes for inclusion in Master

Generally the advice is the same as for the ardupilot flight code ([instructions here](#)) but here is a very short summary of the steps:

- Sign up a member of [Git hub](#)
- Create a personal Fork of the Mission Planner by going to <https://github.com/ArduPilot/MissionPlanner> and click on Fork (Upper right corner area) This creates a copy (fork) of Mission Planner files in your Git Hub account.
- Clone your personal repo (created with the Fork above) to your PC
- Create a new branch in your repo and commit your changes and push these back to GitHub (these will only go into your repo on GitHub).
- Use the GitHub web page to create a Pull Request from your branch
- The owner of Mission Planner (Michael Oborne) will receive an email notifying him of your Pull Request. He will most likely review, provide feedback and if he accepts the commit it will be added to master.

## Building for Erle-Brain 2

These instructions explain how to build ArduPilot on board the Erle-Brain 2.

### Note

Alternatively you can follow [Erle-Robotics's instructions](#) on how to build from source on your PC.

### Connection and setup

Give Erle-Brain Internet access by plugging the Ethernet wire into the RJ-45 connector.

Connect to Erle-Brain 2:

```
$ ssh erle@erle-brain-2.local
```

### Tip

Other connection methods are discussed in the manufacturers [documentation](#)

Clone the source:

```
#Move to home
cd ~/
#Clone the repository in home
git clone https://github.com/erlerobot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

### Build

Build for Copter:

```
cd /home/erle/ardupilot/ArduCopter
make erlebrain2 -j4
```

This will build the firmware for a quadcopter. If you wish to build for another frame type (such as hexacopter) append “-hexa” onto the end of the make command (i.e. make erlebrain2-hexa -j4). The full list of available frames can be found in the [targets.mk](#) file.

### Note

If building for Plane, Rover or Antenna Tracker replace the above “ArduCopter” with “ArduPlane”, “APMrover2” or “AntennaTracker”.

### Move firmware to the executable directory

When you compile Copter (in [~/ardupilot/ArduCopter](#)) the executable **ArduCopter.elf** is created in the same folder. You need to move the executable to home (~/ or `/home/erle/~`) because there is a service in Erle-Brain 2 (called **apm.service**) which automatically launches the autopilot from those locations.

```
#Assuming present working directory is: ~/ardupilot/ArduCopter
sudo cp ArduCopter.elf ~/
```

### Tip

The service mentioned above, launches **ArduCopter** by default, using a bridge for sending telemetry data and GPS placed in **ttyAMA0** by default

You can find additional information about the default launch process and launch configuration options in the [documentation](#).

#### Tip

If you are unable to copy the executable it may be because the destination file is locked. This is because the autopilot is already running. Use the following command to stop the running service:

```
systemctl stop apm.service
```

Now copy the executable in home (~ or `/home/erle/`) and restart the service in order to execute the new executable:

```
systemctl start apm.service
```

Or simply, restart the board once you have copied the new executable

## Building for Erle-Brain

#### Warning

This page is under construction. Links here are to the top level of Erle-Robotics documentation because deeper URLs keep on being broken.

These instructions explain how to build ArduPilot on the Erle-Brain board.

#### Tip

Alternatively you can follow Erle-Robotics's [documentation](#) on how to build from source.

### Connection and setup

Connect to Erle-Brain using microUSB:

```
sudo ifconfig eth0 192.168.7.1
```

```
ssh root@192.168.7.2
```

#### Tip

Check the interface Erle-Brain creates using **ifconfig** command.

Give Erle-Brain Internet access connecting Ethernet wire into RJ45 connector and configure the interface:

```
$ sudo ifconfig eth0 up
$ sudo dhclient eth0
#Check if Erle Brain has Internet access
$ ping www.google.es
# press |ctrl| |c| to exit
```

Clone the source:

```
cd
git clone https://github.com/erlerobot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

## Build

Build for Copter:

```
cd /home/pi/ardupilot/ArduCopter
make pxf -j4
```

This will build the firmware for a quadcopter. If you wish to build for another frame type (such as hexacopter) append “-hexa” onto the end of the make command (i.e. make -j4 pxf-hexa). The full list of available frames can be found in the [targets.mk](#) file.

Note

If building for Plane, Rover or Antenna Tracker replace the above “ArduCopter” with “ArduPlane”, “APMrover2” or “AntennaTracker”.

### Move firmware to the executable directory

Move the executable to the directory from where it is normally started:

```
sudo cp ArduCopter.elf ~/
```

Tip

If you are unable to copy the executable it may be because the destination file is locked because it is already running. Use the following command to stop the running service

```
systemctl stop apm-copter.service
```

Tip

Autopilot launch configuration is covered in the documentation.

## Building for Bebop 2

These instructions explain how to use ArduPilot for the [Bebop2](#) on a Linux machine. The Bebop 2 is based on the same architecture as the Bebop with a few noticeable changes, not the least being a much better quality GPS (UBlox GPS with a bigger antenna).

Warning

Making the changes described in this article will void your warranty! Parrot's technical support will not help you with this hack or to recover your original software.

Warning

Hacking a commercial product is risky! This software is still evolving, and you may well find issues with the vehicle ranging from poor flight to complete software freeze.

That said, it is almost always possible to recover a drone and members of the ardupilot dev team can likely help people hacking or recovering their Bebop on [this google group](#). Prepare to spend some time, patience and develop some hardware/software skills.

### Building ArduCopter for Bebop 2

The instructions are exactly the same as [the one used for Bebop](#)

## Uploading the Firmware

1. Install adb (android debug tool):

```
sudo apt-get install android-tools-adb
```

2. Connect to the Bebop2's WiFi network (BebopDrone-XXXX).

3. Enable adb server by pressing the power button 4 times.

4. Connect to the Bebop's adb server on port 9050:

```
adb connect 192.168.42.1:9050
```

5. If the previous command returns an error, try again (press the power button 4 times and retry).

6. Remount the system partition as writeable:

```
adb shell mount -o remount,rw /
```

7. Push the stripped arducopter binary to the Bebop2:

```
adb push arducopter /usr/bin/
```

## Starting ArduPilot

1. Kill the regular autopilot:

```
kk
```

2. Launch Copter:

```
arducopter -A udp:192.168.42.255:14550:bcast -B /dev/ttyPA1 -C udp:192.168.42.255:14551:bcast -l /data/ftp/internal_000/APM/logs -t /data/ftp/internal_000/APM/terrain
```

## Launch Copter at startup

As for Bebop, modify the init script `/etc/init.d/rcS_mode_default`. Comment the following line:

```
DragonStarter.sh -out2null &
```

Replace it with:

```
arducopter -A udp:192.168.42.255:14550:bcast -B /dev/ttyPA1 -C udp:192.168.42.255:14551:bcast -l /data/ftp/internal_000/APM/logs -t /data/ftp/internal_000/APM/terrain &
```

1. Enable adb server by pressing the power button 4 times.

2. Connect to adb server as described before:

```
adb connect 192.168.42.1:9050
```

3. Re-mount the system partition as writeable:

```
adb shell mount -o remount,rw /
```

4. In order to avoid editing the file manually, you can download [this one](#).

5. Save the original one and push this one to the bebop

```
adb shell cp /etc/init.d/rcS_mode_default /etc/init.d/rcS_mode_default_backup  
adb push rcS_mode_default /etc/init.d/
```

7. Sync and reboot:

```
adb shell sync  
adb shell reboot
```

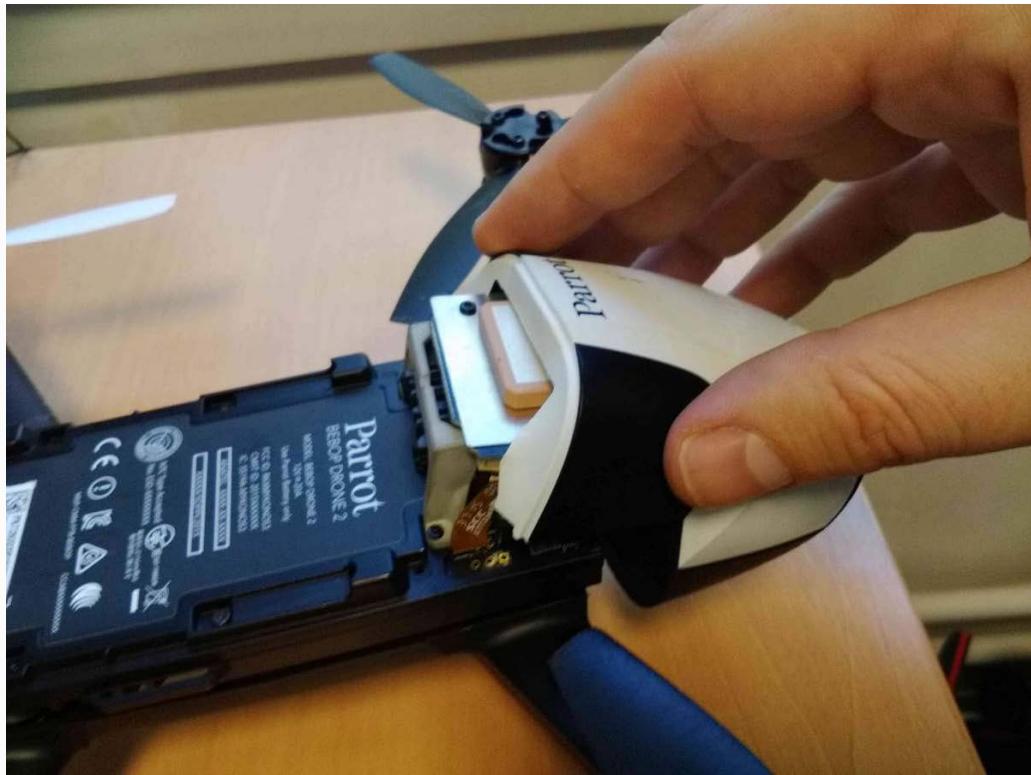
## Recovery

For recovery, you can use the same cable as the one used on Bebop, see [here](#).

1. Remove the two screws using a torx T6 screwdriver



2. Remove the neck by pulling it towards the front of the Bebop



3. The UART connector is located on the right side



4. Plug the cable with the black wire at the front



5. Connect to the bebop with the UART port using any terminal emulator

6. Copy the backup rcS file back to its place

```
mount -o remount,rw /  
cp /etc/init.d/rcS_mode_default_backup /etc/init.d/rcS_mode_default
```

## 7. Sync and reboot

```
sync
reboot
```

### Flying and RC over UDP

Flying and RC over UDP instructions are the same as [the ones for Bebop](#)

### Basic configuration and frame parameters

1. The set of tuning parameters can be found [here](#). These are not yet fully tuned for Bebop 2
2. In order to do the basic configuration and calibration, you can use any of the GCSs and perform:
  1. Magnetometer Calibration
  2. RC Calibration
  3. Accelerometer Calibration

### Additional information

The loiter mode quality is very good compared to the first Bebop because of the (much better) UBlx GPS. It is now safe to takeoff and land in the mode you want.

There is still no support for video yet and the optical flow and sonar are currently under development.

This is a good time to participate and help improve them!

## Building for Qualcomm Snapdragon Flight Kit

This article shows how to build ArduPilot for [Qualcomm® Snapdragon Flight™ Kit \(Developer's Edition\)](#) with *Make* on Linux.

### Warning

Due to some rather unusual licensing terms from Intrinsyc we cannot distribute binaries of ArduPilot (or any program built with the Qualcomm libraries). So you will have to build the firmware yourself.

### Overview

There are two ports of ArduPilot to this board:

- [QFLIGHT](#) runs mostly on the ARM cores, with just sensor and UARTs on the DSPs. This port is much easier to debug and you can use all of the normal Linux development tools.
- [HAL\\_QURT](#) runs primarily on the DSPs, with just a small shim on the ARM cores. This port has better performance due to its extremely accurate realtime scheduling, and is also more robust as it can keep flying if Linux crashes for some reason (the QFLIGHT port relies on both Linux and QURT working to keep flying). However it is harder to debug!

The build instructions for the two ArduPilot ports are almost exactly the same (both are covered here).

### Tip

We recommend that developers use QFLIGHT port for development and debugging, but recompile and target QURT for production use.

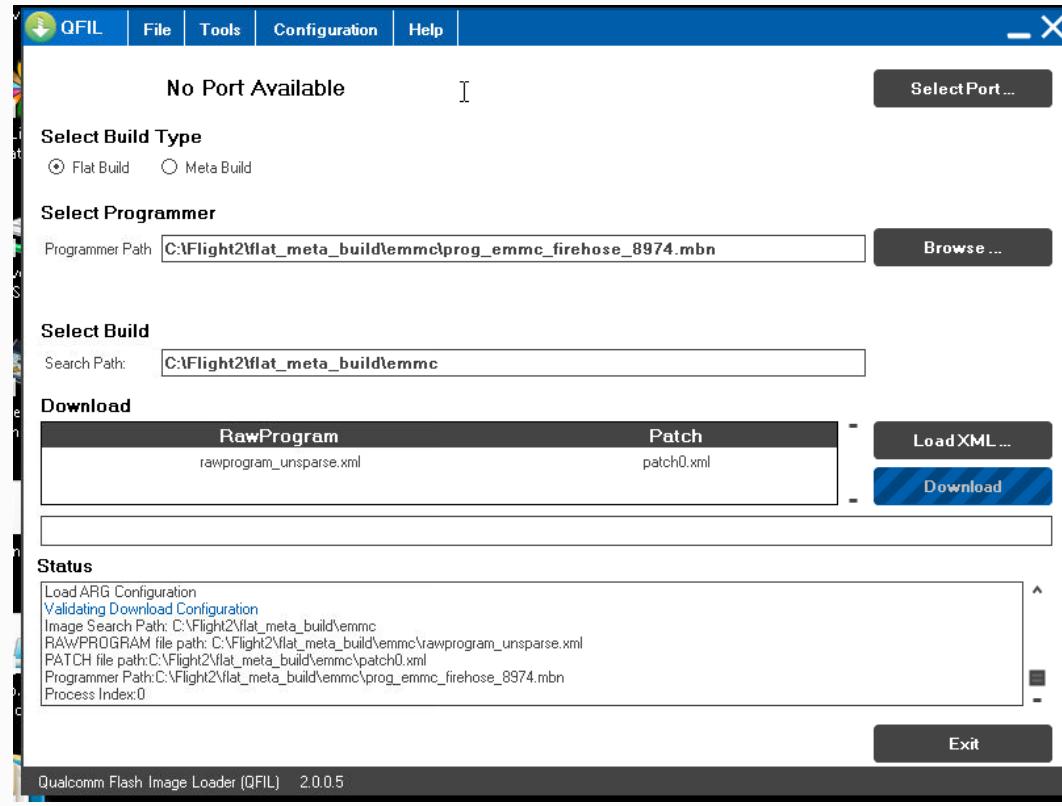
The instructions in this article demonstrate building for Copter (from the ArduCopter directory). Plane and Rover are build in the same way, from their respective vehicle source directories.

## Preparing the board

Flashing the OS with QFIL on Windows¶

Boards purchased from Intrinsic come pre-loaded with an operating system (i.e. Ubuntu) but it may not if you receive a board from a different source. Use the instructions below to flash the OS onto the board from a Windows PC.

1. Download and install QFIL ([here](#) - search for the green download button with "QPST 2.7.422" written below it).
2. Put the board into bootloader mode by first powering the board using the power brick and then connect with a USB cable to your PC (note the board has a USB3.0 port but a normal micro USB cable will work)
3. Start QFIL and the COM port the board is connected to should automatically appear to the left of the **Select Port** button. Normally it appears with the name "QDLoader 9008".
4. Under "Select Programmer" click the **Browse** button and find prog\_emmc\_firehose\_8974.mbn (To-Do: which of the packages was this from?)
5. Under "Download" the rawprogram\_unsparse.xml" should be found automatically (?)
6. Click the **Download** button and if all goes well the OS will be flashed to the board



Connecting to the board using the FTDI cable¶

The board should come with an FTDI cable and small green serial adapter board which can be plugged into the rectangular black connector on the top of the board next to the USB port. Use a ssh program such as Putty to connect to the FTDI cable's serial port at 115200 baud (no password is required to log into the board this way).

Changing the root password and enabling login as root¶

Connect using the FTDI cable using the instructions above and change the root password by typing `passwd` on the command line and entering your desired password twice (ie. "penguin").

Enable logging in as root by editing /etc/shell/sshd\_config and modifying line 28 to look like below:

```
PermitRootLogin yes
```

Connecting to the board's wifi access point¶

The board's wifi access point will appear as ssid "Atlanticus\_XXXXX" (where X is a 5 digit number) and password "password".

The IP address assigned to your computer (default range is 192.168.1.10 ~ 192.168.1.20) can be changed by modifying the files below. This may be necessary to avoid conflicting with your home/business wifi network.

Lines 304, 305 of /etc/dnsmasq.conf as shown below.

```
interface=wlan0
```

```
dhcp-range=192.168.**2**.10,192.168.**2**.20,infinite
```

Line 39 of /etc/network/interfaces.d/qca6234.cfg.softap as shown below.

```
`` address 192.168.2.1``
```

You should be able to use a terminal program such as Putty to ssh to the board at the IP address shown above (i.e. 192.168.2.1) using user/password of linaro/linaro or root/penguin.

Correct home directory permissions¶

By default the /home/linaro directory permission are incorrect.

```
sudo chown -R linaro.linaro /home/linaro
```

Create a logs directory¶

On the flight board, create the logs directory in a location where the DSPs can write to:

```
mkdir /usr/share/data/adsp/logs
```

Checking and upgrading the Ubuntu kernel version on the board¶

Connect to the board using the FTDI cable or ssh via sifi can type "uname -a". The output should appear as below (or with a later date), if it does not then it should be updated.

```
Linux linaro-developer 3.4.0-eagle8074 #1 SMP PREEMPT Wed Dec 9 17:42:13 PST 2015 armv7l armv7l armv7l GNU/Linux
```

To upgrade to a later version of Ubuntu first [download and extract the latest "Flight\\_BSP\\_X.X" file from the Intrinsic site](#) (Note: these files had at least temporarily disappeared as of Jan-2016).

Upgrade using the fastboot-all script:¶

On an Ubuntu machine find the Binaries/fastboot-all.bat script from the above download.

Edit the script and remove the reboot line (line 16) which is shown below (this reboot can cause the board to become bricked if the upgrade fails)

```
fastboot reboot
```

Run the script:

```
sudo ./Binaries/fastboot-all.bat
```

After the upgrade, the board will be completely wiped meaning any previous setup (i.e. root passwords, wifi IP addresses) will need to be redone.

Note: On 25-Jan-2016, while after performing this upgrade and writing this wiki page we found the /firmware/image was out of date. We should add instructions on which files needed to be updated and where the new files can be found.

#### Upgrade by manually copying images [¶](#)

- extract the contents of the above zip and find the following files in the Binaries directory
  - boot.img, cache.img, persist.img, system.img
- on an Ubuntu machine, unpack 3 of the 4 images:
  - simg2img cache.img cache.ext4
  - simg2img persist.img persist.ext4
  - simg2img system.img system.ext4
- transfer these four files into a new /images directory on the flight board (either transfer via wifi or put on an sd card)
  - boot.img, cache.ext4, persist.ext4, system.ext4
- check if any partitions are in use by typing "mount"
  - normally only "persist" will be being used so unmount it with the "umount /mnt/persist/" command
- mount the images:
  - dd if=boot.img of=/dev/disk/by-partlabel/boot bs=1M
  - dd if=cache.ext4 of=/dev/disk/by-partlabel/cache bs=1M
  - dd if=persist.ext4 of=/dev/disk/by-partlabel/persist bs=1M
  - dd if=system.ext4 of=/dev/disk/by-partlabel/boot bs=1M

#### Install the baro and mpu9250 drivers [¶](#)

Copy these two files to the flight board's /usr/share/data/adsp:

libbmp280.so, libmpu9x50.so

#### Edit dangerous Q6 service startup script [¶](#)

On the flight board, edit /etc/init/q6.conf script and comment out the line below which, if left in place, can cause the board to stall forever during the boot up process is "q6" fails to start

```
#watch -n 1 --precise -g grep -m 1 "2" /sys/kernel/debug/msm_subsys/adsp && true
```

#### Putting the board in to "Storage mode" [¶](#)

During the upgrade process, if the above step is skipped, it is possible to get the board into a state where it will not completely boot up. You will be unable to login using FTDI or wifi. If this occurs you can exploit a race condition in the startup sequence to get the board in "storage mode" which allows accessing the disks on the board.

- first ensure both power and usb cables are disconnected
- plug in the power
- quickly plug in the usb
- disconnect the power brick, hopefully the led will turn red
- plug in the power

If all goes well, 10 or 20 disk devices will appear on the Windows or Ubuntu machine connected via USB. The files on the disk can be edited including perhaps editing the startup scripts to resolve the boot-up issue.

### **Preconditions for building**

These instructions will only work for 64bit Linux machines (including Ubuntu).

Get the source code [¶](#)

First clone the source:

```
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

Get additional tools/libraries¶

To build ArduPilot for either port you will also need 3 library packages from Intrinsic (download links are supplied when you buy the board):

- **HEXAGON\_Tools** package, tested with version 7.2.11
- **Hexagon\_SDK** packet, version 2.0
- **HexagonFCAddon** package, tested with Flight\_BSP\_1.1\_ES3\_003.2

These packages should all be unpacked in a `$HOME/Qualcomm` directory.

## **Building for QURT**

To build Copter you do:

```
cd ArduCopter
make qurt -j4
```

Upload the firmware to the board by joining to the WiFi network of the board and entering the following command (where `myboard` is the hostname or IP address of your board):

```
make qurt_send FLIGHT_BOARD=myboard
```

This will install two files:

```
/root/ArduCopter.elf
/usr/share/data/adsp/libardupilot_skel.so
```

To start ArduPilot just run the `.elf` file as root on the flight board.

```
/root/ArduCopter.elf
```

### Note

For the QURT port you can't use arguments to specify the purpose of each UART.

By default ArduPilot will send telemetry on UDP 14550 to the local WiFi network. Just open your favourite MAVLink compatible GCS and connect with UDP.

## **Building for QFLIGHT**

To build Copter for QFLIGHT do:

```
cd ArduCopter
make qflight -j4
```

Upload the firmware to the board by joining to the WiFi network of the board and entering the following command (where `myboard` is the hostname or IP address of your board):

```
make qflight_send FLIGHT_BOARD=myboard
```

This will install two files:

```
/root/ArduCopter.elf  
/usr/share/data/adsp/libqflight_skel.so
```

To start ArduPilot just run the **.elf** file as root on the flight board. You can control UART output with command line options. A typical startup command would be:

```
/root/ArduCopter.elf -A udp:192.168.1.255:14550:bcast -e /dev/tty-3 -B qflight:/dev/tty-2 --dsm /dev/tty-4
```

That will start ArduPilot with telemetry over UDP on port 14550, GPS on tty-2 on the DSPs, Skektrum satellite RC input on tty-4 and ESC output on tty-3.

By default ArduPilot will send telemetry on UDP 14550 to the local WiFi network. Just open your favourite MAVLink compatible GCS and connect with UDP.

### **Starting ArduPilot on boot**

You can also set up ArduPilot to start on boot by adding the startup command to **/etc/rc.local**. For example, on QURT build you'd add the line:

```
/root/ArduCopter.elf &
```

## **Building for Bebop on Linux**

These instructions clarify how to build ArduPilot for the [Bebop](#) flight controller board on a Linux machine. More details on the Bebop can be found [here](#).

Tip

The instructions for running ArduPilot on [Bebop 2](#) can be found [here](#).

Warning

Making the changes described in this article will void your warranty! Parrot's technical support will not help you with this hack or to recover your original software.

Warning

Hacking a commercial product is risky! This software is still evolving, and you may well find issues with the vehicle ranging from poor flight to complete software freeze.

That said, it is almost always possible to recover a drone and members of the ardupilot dev team can likely help people hacking or recovering their Bebop on [this google group](#). Prepare to spend some time, patience and develop some hardware/software skills.

### **Upgrading the firmware**

As of Nov 2015, the Bebop ships with a version of Linux that cannot run ArduPilot and must be upgraded. In order to upgrade it, you will need to download a custom version [here](#).

In order to upgrade to this version:

1. Power up your Bebop
2. Connect to its Wi-Fi network (BebopDrone-XXXX)
3. Connect to it via ftp

```
ftp 192.168.42.1
```

4. go to the eMMC directory

```
cd internal_000
```

5. Upload the update file

```
put bebopdrone_update.plf
```

6. Connect to the Bebop by telnet

```
telnet 192.168.42.1
```

7. Sync and reboot

```
sync  
reboot
```

8. Wait for the Bebop to perform the update (this could take several minutes)

#### Note

Don't shutdown your Bebop during this time

9. When the update is complete you can connect again via Wi-Fi and telnet and verify the update by checking the software version indicates 0.0.0 (not an official release)

```
cat version.txt
```

## Build ArduCopter for Bebop

### Tip

You can skip this step if you just want to try out the (experimental) binary version.

The following steps show how to build a custom version of the Copter software for Bebop:

Install armhf toolchain [¶](#)

On Ubuntu from 12.04 [¶](#)

1. Install the official arm-linux-gnueabihf toolchain

```
sudo apt-get install gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf
```

On other Linux distributions [¶](#)

1. Install the *arm-gnueabihf* tool chain that can be downloaded from [here](#)

2. Extract the tar archive (for instance in /opt)

```
sudo tar -xjvf gcc-linaro-arm-linux-gnueabihf-4.9-2014.07_linux.tar.bz2 -C /opt/
```

3. Add the path to the toolchain to the PATH variable

```
export PATH=/opt/gcc-linaro-arm-linux-gnueabihf-4.9-2014.07_linux/bin:$PATH
```

Download and compile ArduCopter¶

1. You need to install git first (see [instructions here](#))
2. Clone ardupilot repository

```
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

3. Building the flight control firmware is nearly identical for [building for the Pixhawk](#) except the `make` command is:

```
cd ArduCopter
make bebop
```

5. Strip the binary to reduce the memory footprint:

```
arm-linux-gnueabihf-strip ArduCopter.elf -o arducopter
```

## Uploading the firmware

1. If you haven't built the firmware as described in the previous steps you can download a binary version [here](#)
2. Connect again by ftp and go to the eMMC directory
3. Put the arducopter binary

```
put arducopter
```

4. Connect to the Bebop via telnet
5. Copy arducopter to /usr/bin and change permissions

```
cp /data/ftp/internal_000/arducopter /usr/bin
chmod +x /usr/bin/arducopter
```

## Starting ArduPilot

1. Connect via telnet
2. Kill the regular autopilot

```
kk
```

3. Launch Copter

```
arducopter -A udp:192.168.42.255:14550:bcast -B /dev/ttyPA1 -C udp:192.168.42.255:14551:bcast -l
/data/ftp/internal_000/APM/logs -t /data/ftp/internal_000/APM/terrain
```

## Changing the GPS configuration

In order to get Bebop's GPS to send the NMEA frames that APM's NMEA driver understands, you need to change its configuration. To achieve this you will need to stop the in-build autopilot as described

previously (and don't launch Copter yet):

1. Download the **gps\_config** file [here](#)
2. Connect to the Bebop via ftp and go to the eMMC directory as indicated in the “Upgrading the firmware” section above
3. Put the config file

```
put gps_config.txt
```

4. Connect to the Bebop via telnet
5. Copy **gps\_config.txt** in /etc/

```
cp /data/ftp/internal_000/gps_config.txt /etc/
```

6. Launch the GPS config updater

```
libgps_cli
```

7. Wait for NMEA messages to be displayed in the console
8. Stop **libgps\_cli** by typing **Ctrl-C**

### **Launch Copter at startup**

It is a lot more convenient to automatically execute Copter startup than connect and do this manually. In order to do so, the startup scripts need to be hacked in the following way.

#### Warning

This part is critical since you have to edit the startup script. If you do something wrong here, you could end up with a Bebop that can no longer boot properly. If this happens you will have to get a UART cable to recover.

The startup script is located at **/etc/init.d/rcS**. You will need to edit it to remove the lines launching the regular autopilot and replace them by launching Copter. The line in question is the following:

```
DragonStarter.sh -out2null &
```

Replace this with:

```
arducopter -A udp:192.168.42.255:14550:bcast -B /dev/ttyPA1 -C udp:192.168.42.255:14551:bcast -l /data/ftp/internal_000/APM/logs -t data/ftp/internal_000/APM/terrain &
```

In order to avoid editing the file manually, download [this](#) rcS file.

1. Make a copy of the original rcS file for recovery purpose

```
cp /etc/init.d/rcS /etc/rcS_backup
```

2. Connect to the Bebop via ftp and put the rcS file in the eMMC as described before for the other files.
3. Then copy it manually to overwrite **/etc/init.d/rcS** and change permissions

```
cp /data/ftp/internal_000/rcS /etc/init.d/rcS
chmod +x /etc/init.d/rcS
```

#### 4. Sync and reboot

```
sync
reboot
```

#### 5. In case you want to put your Bebop back to normal and use the normal autopilot and app again, just replace `/etc/init.d/rcS` with the backup file, sync and reboot

```
cp /etc/rcS_backup /etc/init.d/rcS
sync
reboot
```

#### Note

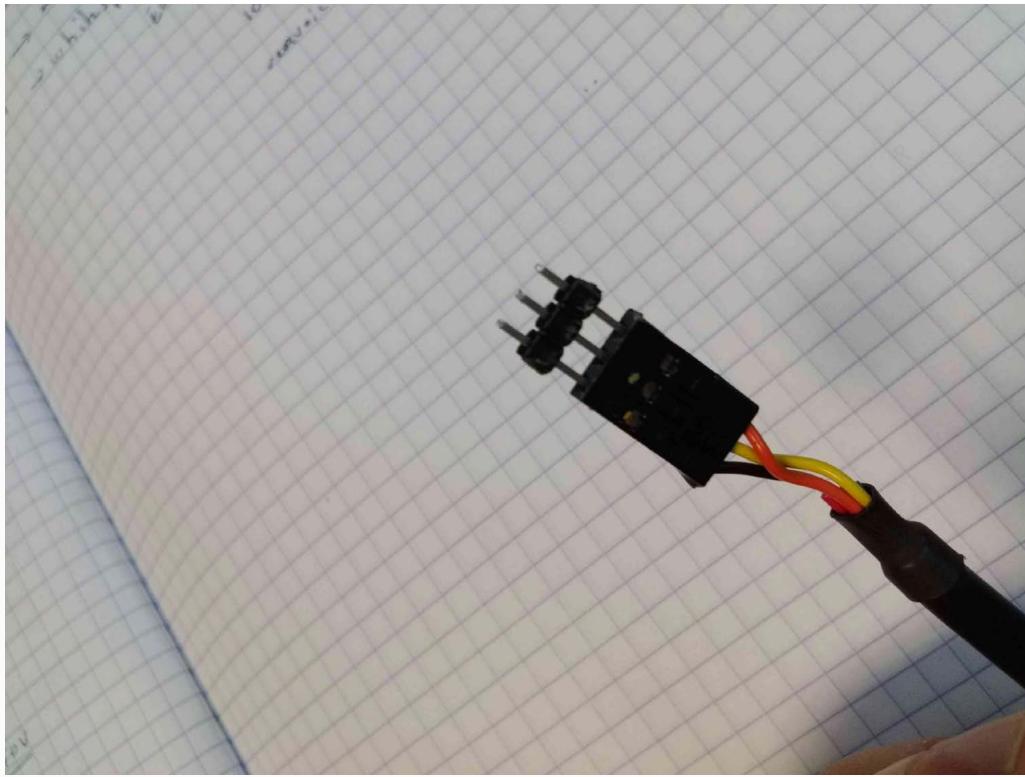
If you put your software back to normal and use your Bebop with FreeFlight smartphone App, you might be asked to upgrade your software version. If you do so, you will have to repeat some of the previous steps, at least for the GPS config, copying arducopter and modifying the init scripts. Regarding the need to upgrade to a custom version, it will depend on whether some options will or won't be available in the following release. Informations to follow...

#### **Recovery**

1. In case something went wrong and you are not able to boot your Bebop anymore
2. The UART port is located under the Bebop's neck on the right side (facing the front camera)



3. You will have to pull back the polystyrene a bit but it shouldn't cause much damage
4. Get a UART cable like [this one](#) or any FTDI 3 pin cable (GND TXD RXD)
5. Get headers like [these ones](#) and plug them into the cable like this:



Note

The color codes for the cable are usually:

- black = GND
- yellow = RXD
- orange = TXD

6. Plug the cable into the Bebop like this:



Note

Be careful about the pinout:

- black: front

- yellow: middle
  - orange: back
7. Install a UART terminal emulator like minicom and connect to a Bebop once it is powered up
  8. Copy the backup rcS file back to its original place, sync and reboot:

```
mount -o remount,rw /
cp /etc/rcS_backup /etc/init.d/rcS
sync
reboot
```

## Flying

FreeFlight 3 is not compatible with ArduPilot and you will therefore have to use [one of the supported GCS](#). Connect to the Bebop via Wi-Fi and just start your GCS, it should connect automatically if you setup the link to UDP (in case it is needed).

The [SkyController](#) is not compatible with apm with its regular firmware. You would need to flash an alternative version in order to be able to control your Bebop with it (information about that is coming soon...).

In order to pilot the Bebop manually, Mission Planner GCS users can use a [gamepad as described here](#). Alternatively use the RCOOutput UDP interface on port 777 on the Bebop, with a Linux PC (or board type Raspberry Pi) and a USB gamepad.

## Controlling the Bebop via RC over UDP on Linux

1. In order to control the arducopter for Bebop via RC over UDP, you can either write an application using [this protocol](#) and sending a packet every 10ms
2. Or use [joystick\\_remote](#) Linux application
3. In order to do so, clone the git repository:

```
git clone https://github.com/jberaud/joystick_remote.git
```

4. Build it

```
cd joystick_remote
make
```

5. Plug a USB gamepad (the list of supported gamepads is explained if you type joystick\_remote –help)
6. In case your gamepad is not supported you can easily add support for it if you know its mapping
7. Connect to the Bebop via Wi-Fi and launch the application:

```
./joystick_remote -d /dev/input/js[X] -t [gamepad] -r 192.168.42.1:777
```

where [X] is the device number of your joystick that you can easily find, usually 0 but sometimes 1 if your laptop already includes an input device like an accelerometer and [gamepad] is one of the supported gamepads.

8. so for an XBox 360 gamepad mapped on /dev/input/js0 the command line becomes

```
./joystick_remote -d/dev/input/js0 -t xbox_360 -r 192.168.42.1:777
```

9. The flight modes have to be set in Copter's parameters in order to use the buttons to set the flight

modes

### Basic configuration and frame parameters

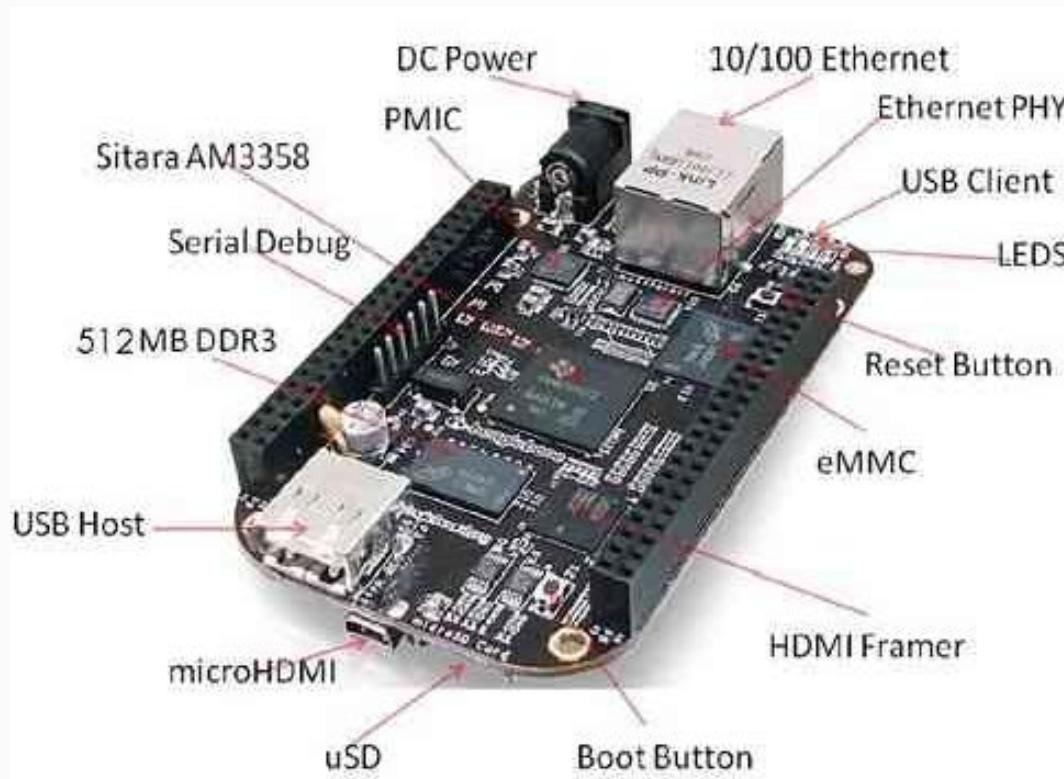
1. In order to do the basic configuration and calibration, you can use any of the GCSs and perform
  - Magnetometer Calibration
  - RC Calibration
  - Accelerometer Calibration
2. Thanks to Leonard Hall, we have a very good set of tuning parameters that you can find [here](#)

### Known limitations

- The GPS of the Bebop isn't very good compared to a UBlox GPS and therefore the Bebop drifts significantly in Loiter, PosHold and other GPS modes
- Mission run in Auto mode work reasonably well but we recommend you takeoff and land in a non-GPS mode such as AltHold or Stabilize.
- Some work will be done to improve support for this GPS
- The optical flow is currently under development
- There is currently no support for video streaming and capture

### Building ArduPilot for BeagleBone Black on Linux

#### Get your BeagleBone running Debian



#### **BBB title picture**

You will need a microSD card of 4GB or greater and a 5VDC power supply of 1A or higher. You can also use a powered USB hub. It is highly recommended that you not use the USB port on a laptop, which has current limitations.

Go to the BeagleBoard website and get the [latest image](#). You may find multiple images on the page. For

ArduPilot on BeagleBone Black you have to download the **Debian (BeagleBone Black - 2GB eMMC)** image. You can find it under **BeagleBone Black (eMMC flasher)** header.

For example, the current image is **BBB-eMMC-flasher-debian-7.5-2014-05-14-2gb**. The date (in this case 2014-05-14) and the version number (in this case 7.5) may change. Make sure you download the latest image.

On Ubuntu/Mac OS X¶

Verify Image with:

```
md5sum BBB-eMMC-flasher-debian-7.5-2014-05-14-2gb.img.xz
74615fb680af8f252c034d3807c9b4ae  BBB-eMMC-flasher-debian-7.5-2014-05-14-2gb.img.xz
```

before plugging your SD card into your computer, type

```
df -h
```

this will list the current mounted drives.

Now plug in your SD card and type

```
df -h
```

and see what the drive that was added is called.

In my case it was /dev/sdd1

in which case in the command below, the it is written dd of=/dev/sdd replace this with the correct address for your SD card.

Warning

IF YOU GET THIS WRONG YOU CAN WIPE YOUR HDD  
then xzcat it to your SD card in your Ubuntu/Mac OS X machine

```
xzcat BBB-eMMC-flasher-debian-7.1-2013-10-08.img.xz | dd of=/dev/sdd bs=1M
```

when this is finished, remove the SD Card and place it into your BBB.

On Windows¶

Download Win32 disk imager from [here](#). Insert your SD card, then start the application that you downloaded.

Select the image that you downloaded and then press **Write**.

when this is finished, remove the SD Card and place it into your BBB.

Flashing the image to the eMMC¶

Place the SD card in your BBB. Make sure you have removed Ethernet and all other USB devices from your BBB. Connect it to a wall adapter (5VDC 1A) or a powered USB hub.

Press the boot button, and hold it down while booting, until all four blue lights are solid.

Let go, and wait. It will take around 15 mins.

The lights will flicker a lot at this stage, you will know it is complete when the four lights return to solid Blue (no flickering). Power down the BBB and remove the SD card.

Connect the BeagleBone Black to your machine using the USB cord that's provided with it. Depending on your OS, install the required [drivers](#).

Now ssh into the BeagleBone Black by typing

```
ssh root@192.168.7.2
```

There is no password for the root user on the Debian image

You could also connect the BBB to your local network over Ethernet and ssh to it.

## **The PixHawk Fire Cape**

In order to run APM in the BeagleBone black you'll need to use the right set of sensors. Most of these sensors are included in the PixHawk Fire cape (PXF), an open hardware board available from [Erle Robotics store](#).

## **Making the rt kernel**

(taken from [http://wiki.beyondlogic.org/index.php/BeagleBoneBlack\\_Building\\_Kernel](http://wiki.beyondlogic.org/index.php/BeagleBoneBlack_Building_Kernel))

modified for the RT version, and to simplify.

### Compiling the BeagleBone Black Kernel

The following contains instructions for building the BeagleBone Black kernel on your PC with Ubuntu 13.04 O.S.

#### **to make it simple, run**

```
sudo su
```

you may need to put in your password here...

if you do not have this already, make the following directory

```
mkdir /home/YOUR_USERNAME/export
```

mkdir /home/YOUR\_USERNAME/export/rootfs

## **Prerequisites**

### ARM Cross Compiler

To compile the linux kernel for the BeagleBone Black, you must first have an ARM cross compiler installed on your linux box. I use gcc-4.7-arm-linux-gnueabi-base that comes with Ubuntu 13.04. To install the compiler run:

```
apt-get install gcc-arm-linux-gnueabi
```

### GIT

The Beaglebone patches and build scripts are stored in a git repository. Install git:

```
apt-get install git
```

And configure with your identity.

```
git config --global user.email "your.email@here.com"
```

### Izop Compression¶

The kernel is compressed using Izop. Install the Izop parallel file compressor:

```
apt-get install lzop
```

### uBoot mkimage¶

The bootloader used on the BeagleBone Black is **u-boot**. u-boot has a special image format called ulimage. It includes parameters such as descriptions, the machine/architecture type, compression type, load address, checksums etc. To make these images, you need to have a mkimage tool that comes part of the u-Boot distribution. Download u-boot, make and install the u-boot tools:

```
wget ftp://ftp.denx.de/pub/u-boot/u-boot-latest.tar.bz2
tar -xjf u-boot-latest.tar.bz2
```

```
cd u-boot-2013.10 (look to see what this is called, it may have changed)
```

```
make tools (don't work with last revision of u-boot need a revision)
install tools/mkimage /usr/local/bin
```

### Compiling the BeagleBone Black Kernel¶

Here we compile the BeagleBone Black Kernel, and generate an ulimage file with the DTB blob appended to the kernel for ease of use.

```
git clone git://github.com/beagleboard/kernel.git
cd kernel
git checkout 3.8-rt
./patch.sh
cp configs/beaglebone kernel/arch/arm/configs/beaglebone_defconfig
wget http://arago-project.org/git/projects/?p=am33x-cm3.git;a=blob_plain\;f=bin/am335x-pm-firmware.bin\;hb=HEAD -O kernel/firmware/am335x-pm-firmware.bin
cd kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- beaglebone_defconfig -j4
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage dtbs -j4
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage-dtb.am335x-boneblack -j4
```

Now we build any kernel modules:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules -j4
```

And if you have your rootfs ready, you can install them:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- INSTALL_MOD_PATH=/home/YOUR_USERNAME/export/rootfs
modules_install
```

## Installing the RT kernel

After you have made the Linux kernel...

ensure you have Debian installed on the beaglebone and ssh into the Beaglebone from Linux....

```
ssh root@192.168.1.3
```

(my ip address, adjust for your beaglebone)

Go to folder /boot/uboot/

```
cd /boot/uboot/
```

make sure there is a backup folder there. If not:

```
mkdir backup
```

then backup your zImage

```
cp zImage uInitrd backup/
```

then

```
ls /lib/modules
```

it should show 3.8.13-bone28 or similar.

Now we need to go to our Ubuntu computers terminal. Ggo to your export folder that you made:

```
cd /home/YOUR_USER_NAME/export/rootfs/lib/modules
```

and run

```
rsync -avz 3* root@192.168.1.3:/lib/modules/
```

then run

```
rsync /home/proficnc/u-boot-2013.10/kernel/kernel/arch/arm/boot/zImage 192.168.1.3 :/boot/uboot/
```

back on your Beaglebone run the following

```
ls /lib/modules
```

you should now have both the old file and the new rt folder.

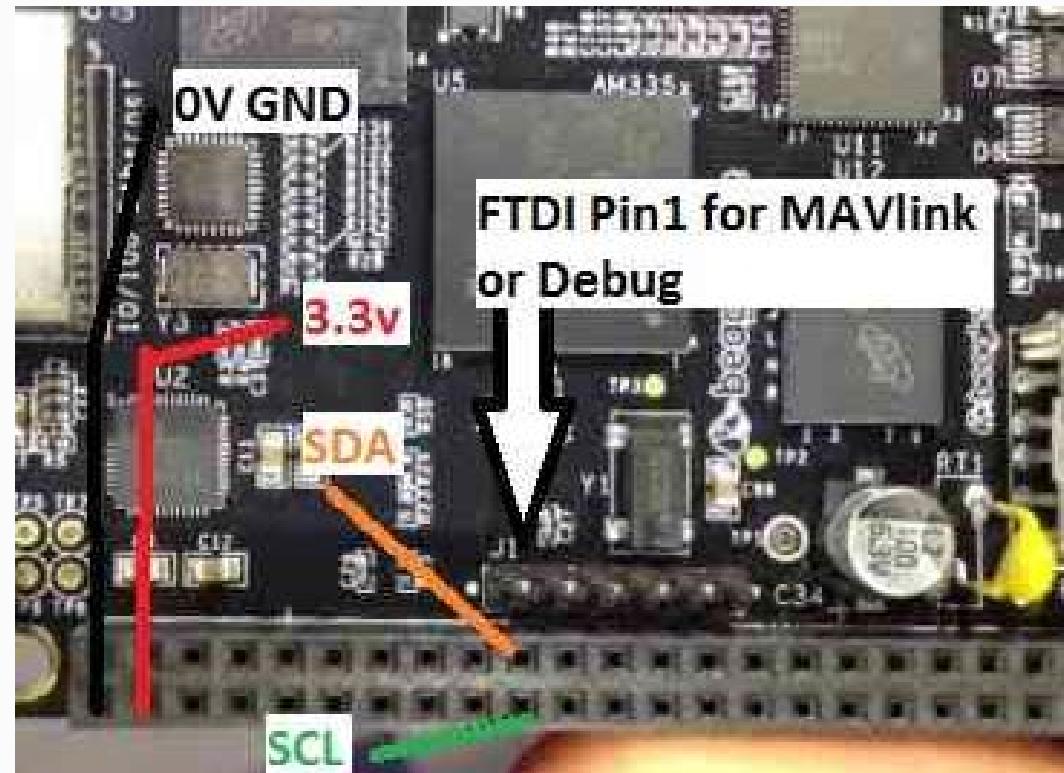
Now type:

```
sync  
reboot
```

Some useful tips

**Hooking up the sensors**

When hooking up your Sensor board it connects as follows (using SHORT wires)

**i2c connection****i2c Debug**

To detect if the i2c is working, you can use the following command

```
i2cdetect -r 1
```

```
coloc.h:274:1: warning: cast from 'const char*' to 'const double*' increases required alignment of target type [-Wcast-align]
** libraries/AP_HAL/utility/ftoa_engine.o
** libraries/AP_HAL/utility/Print.o
** libraries/AP_HAL/utility/print_vprintf.o
** libraries/AP_HAL_AVR/utility/ISRRRegistry.o
** libraries/AP_HAL_FLYMAPLE/utility/EEPROM.o
Building /tmp/ArduPlane.build/ArduPlane.elf
# ArduPlane.elf
root@arm:/~/ardupilot/ArduPlane# i2cdetect -r 1
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-1 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
      0  1  2  3  4  5  6  7  8  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- 53 00 00 00 00 00 00 00 00 00 00 00
60: -- -- -- -- -- -- -- -- 69 -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
root@arm:/~/ardupilot/ArduPlane#
```

## **i2c check**

The numbers: 1e, 53, 69, 77 are the MAG, Gyro, Accel, and Baro.(not in that order)

### **Hooking up your GPS**

To be added

### **Hooking up your Receiver**

To be added

### **Hooking up your servos**

To be added

### **Hooking up your Airspeed sensor**

To be added

### **Devices tested so far**

Responded with the Who Am I request on SPI

1. MPU6000
2. MPU9250 (may have compass issues due to soldering of jumper wire)
3. MS5611 (SPI)

Not responding on SPI

1 LSM9DS0 ( soldering issue, no connection to I/O

Responded to I2C detect

1. CapeID EEPROM 0x54h AT24CS32
2. CapeID COA OTP 0x5Ch AT24CS32
3. Crypto 0x64h ATSHA204
4. Airspeed 0x28h MS4525DO-DS3AIXXXDS
5. Compass Ext 0x1eh HMC5883L
6. Power management 0x24 TPS65217C
7. on-board EEPROM 0x50h unknown
8. HDMI core.... unused, do not enable 0x34

Not responded to I2C test

1. MS5611 (I2C) 0x76h
2. RGB LED Driver 0x55h TCA62724 (is conflicting with non existent Cape eeprom)

### **Adjusting the BBB clock**

`cpufreq-info` shows your current frequency

```

70: -- -- -- -- -- 77
root@arm:/ardupilot/ArduPlane# /tmp/ArduPlane.build/ArduPlane.elf

[855929.903518] hrtimer: interrupt took 61167 ns
^C
root@arm:/ardupilot/ArduPlane# cpufreq-info
cpufrequtils: 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: generic_cpu0
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 300 us.
  hardware limits: 300 MHz - 1000 MHz
  available frequency steps: 300 MHz, 600 MHz, 800 MHz, 1000 MHz
  available cpufreq governors: conservative, ondemand, userspace, powersave, performance
  current policy: frequency should be within 300 MHz and 1000 MHz.
    The governor "ondemand" may decide which speed to use
    within this range.
  current CPU frequency is 300 MHz (asserted by call to hardware).
  cpufreq stats: 300 MHz:nan%, 600 MHz:nan%, 800 MHz:nan%, 1000 MHz:nan%
root@arm:/ardupilot/ArduPlane# ^C
root@arm:/ardupilot/ArduPlane#

```

## clock check

Edit /etc/default/cpufrequtils (you might need to create it if it doesn't exist). Specify the governor with the GOVERNOR variable:

```
nano /etc/default/cpufrequtils
```

add the following.....

```
# valid values: userspace conservative powersave ondemand performance
# get them from cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

```
GOVERNOR="performance"
```

CTRL-X to exit

Y to save

Reboot, and check to see that it has worked

## Installing and Making ArduPilot on BBB

install git, make, gawk, g++, arduino-core on your BBB

```
apt-get install git make gawk g++ arduino-core
git clone git://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

then open

```
cd ArduPlane
```

or

```
cd ardupilot/ArduCopter
```

or

```
cd ardupilot/APMRover2
```

or

```
cd ardupilot/AntennaTracker
```

then

```
make linux
```

from this directory, run the tmp/Plane.elf (or Copter, or Rover)

```
tmp/Plane.elf
```

## Connecting to GCS

To be added.....

## Status

The following table summarizes the *driver development status*:

Milestone	Status
ArduPilot running in the BBB (I2C connected sensors)	Ok
Device Tree for the PXF	WIP
MPU6000 SPI userspace driver	Ok
MPU9150 I2C userspace driver	Ok
LSM9DS0 SPI userspace driver	Coded
MPU9250 SPI userspace driver	Coded
MS5611 I2C/SPI userspace driver	Coded
GPIO userspace driver	WIP
I2CDriver multi-bus aware	WIP
AP_InertialSensor_Linux	ToDo
PRU PWM driver	Ok ( <a href="#">issue</a> with the PREEMPT_RT kernel)
MPU6000 SPI kernel driver	WIP
MPU9150 I2C kernel driver	ToDo
LSM9DS0 SPI kernel driver	ToDo
MPU9250 SPI kernel driver	ToDo
MS5611 I2C/SPI kernel driver	ToDo

Status: ``Ok'', ``Coded'' (needs test), ``WIP'' (work in progress), ``Issue'', ``ToDo''

## Building ArduPilot for Flymaple on Linux

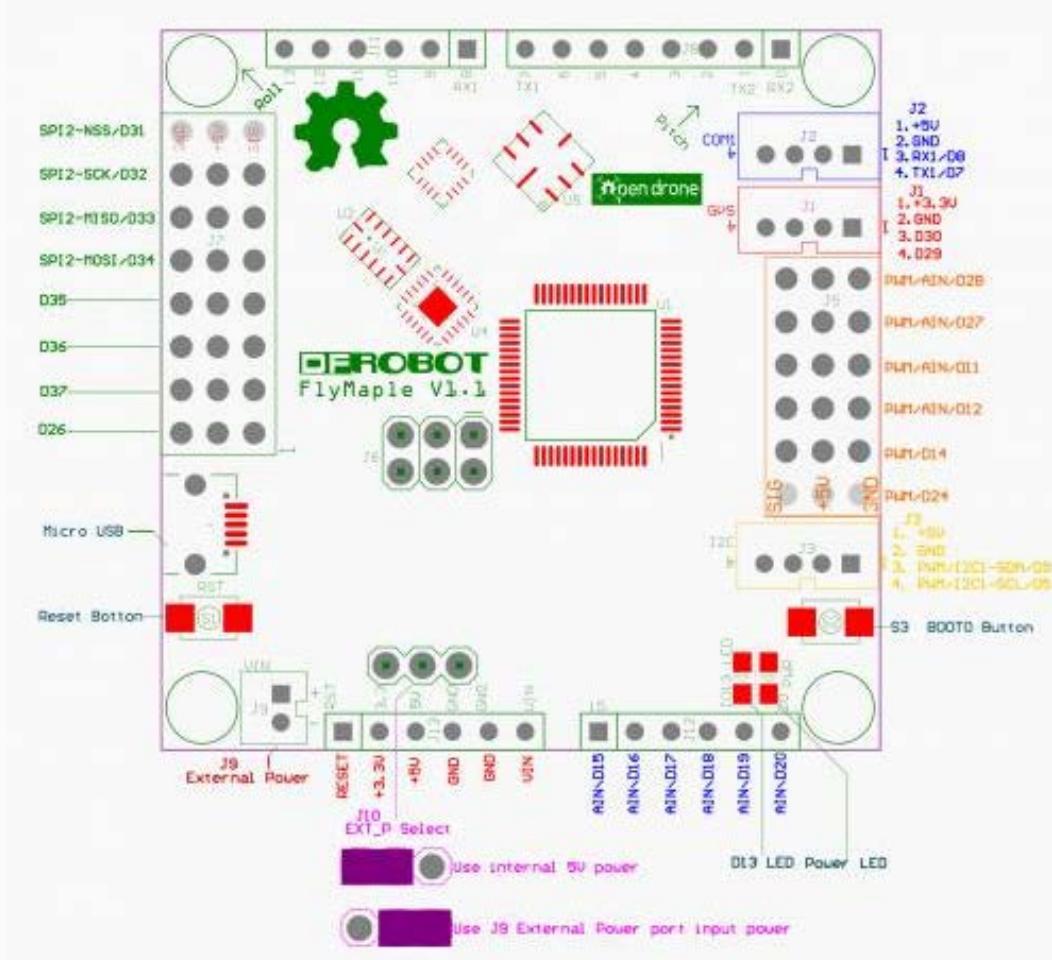
### Note

Support for FlyMaple in ArduPilot was removed in May 2016. This documentation is for older versions

### About Flymaple

- The APMPILOT firmware has been ported to run on [Flymaple](#)
- Flymaple is an inexpensive board based on a 75MHz ARM Cortex-M3 processor.
- It includes 10DOF sensors (accelerometer, gyroscope, magnetometer and barometer).





### How to build APM for Flymaple on Linux.

- You need a number of additional resources to build ardupilot for Flymaple.
- I have assumed that you will install them in your home directory.
- But they can really go anywhere provided you make the appropriate changes to PATH and config.mk

```
cd ~
git clone https://github.com/mikemccauley/libmaple.git
cd libmaple
wget http://static.leaflabs.com/pub/codesourcery/gcc-arm-none-eabi-latest-linux32.tar.gz
tar xvzf gcc-arm-none-eabi-latest-linux32.tar.gz
export PATH=$PATH:~/libmaple/arm/bin
cp main.cpp.example main.cpp
make
```

- At this stage you can test your flymaple CPU and the upload process with 'make install'
- This will upload a simple LED blinking program to your Flymaple board.
- Now download ardupilot APM:

```
cd ~
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

- Now edit config.mk to be something like this:

```
#config.mk START
# Select maple_RET6 for Flymaple
```

```

BOARD = maple_RET6
# HAL_BOARD determines default HAL target.
HAL_BOARD ?= HAL_BOARD_FLYMAPLE
# The communication port used to communicate with the Flymaple
PORT = /dev/ttyACM0
# You must provide the path to the libmaple library directory:
LIBMAPLE_PATH = $(HOME)/libmaple
# Also, the ARM compiler tools MUST be in your current PATH like:
# export PATH=$PATH:~/libmaple/arm/bin
#config.mk END

```

- Now build APM for, say a rover:

```

cd APMrover2
make flymaple
make upload

```

Documentation on how to wire up and configure APM on Flymaple for a buggy type rover are at:

## Git Submodules

This page describes how we use [git submodules](#) in the ArduPilot build. Submodules are used to manage external dependencies in the ArduPilot build, particularly for building for PX4 targets.

### Background

*Git submodules* allow us to automatically bring in dependent git trees in the ArduPilot build. It replaces our old mechanism of having to separately clone the **ArduPilot/PX4Firmware** and **ArduPilot/PX4NuttX** tree when developers want to build for PX4 targets.

#### Note

For now *git submodules* are only used for the PX4 builds. It is likely we will start to use submodules for other builds in the future (for example, we will probably use them for the mavlink message XML files) See <https://git-scm.com/book/en/v2/Git-Tools-Submodules> for more information on *git submodules*.

### Submodule approach

The approach we have implemented in ArduPilot is to use a single level of *git submodules*, with all modules stored in the **modules/** directory. This approach was chosen as it makes for diagnosis of issues with submodules somewhat simpler.

This means that the submodules from the upstream PX4Firmware tree are not used. Instead each required submodule is added as a direct submodule of the ArduPilot tree.

You may also note that the URLs used for the submodules use the old `git://` protocol. This was done to make it less likely we will get accidental commits on the master repositories while developers are getting used to *git submodules* (as the `git://` protocol is read-only). Developers with commit access to the submodules should add a new ardupilot remote with a writeable protocol as needed.

### Common errors

The following is a list of common errors and how to deal with them.

#### Errors with config.mk¶

If you have an existing config.mk you may get an error like this:

```
..mk/px4_targets.mk:8: *** NUTTX_SRC found in config.mk - Please see
http://dev.ardupilot.com/wiki/git-submodules/. Stop.
```

That happens because you have previously built with an external PX4Firmware and PX4NuttX tree, and you need to convert to using submodules. The simplest way to fix this is to remove your **config.mk** file as it is no longer needed. If you want to keep the file for some reason then you can either comment out or removed the lines in **config.mk** which specify the **PX4\_ROOT**, **NUTTX\_SRC** and **UAVCAN\_DIR** variables.

#### Errors from old PX4Firmware and PX4NuttX trees

You may get warnings like these in your build:

```
..mk/px4_targets.mk:23: *** You have an old PX4Firmware tree - see
http://dev.ardupilot.com/wiki/git-submodules/
..mk/px4_targets.mk:26: *** You have an old PX4NuttX tree - see
http://dev.ardupilot.com/wiki/git-submodules/
..mk/px4_targets.mk:29: *** You have an old uavcan tree - see
http://dev.ardupilot.com/wiki/git-submodules/
```

This indicates that you have old PX4Firmware or PX4NuttX directories in ..PX4Firmware or ..PX4NuttX. The warning is harmless and won't prevent you from building. The warning is there so you know that commits and changes made in those directories won't be used.

#### Updating submodules

If you need to manually update submodules you should run the command

```
git submodule update
```

from the root of the ardupilot tree. That will check all submodules for updates in the ardupilot repository and will pull in changes as needed.

#### Disaster recovery

If things have gone very badly wrong with your git tree the simplest thing to do it to remove the modules/ directory completely. Then do a new px4 build with

```
make px4-v2
```

and the submodules will be automatically reinitialised and updated.

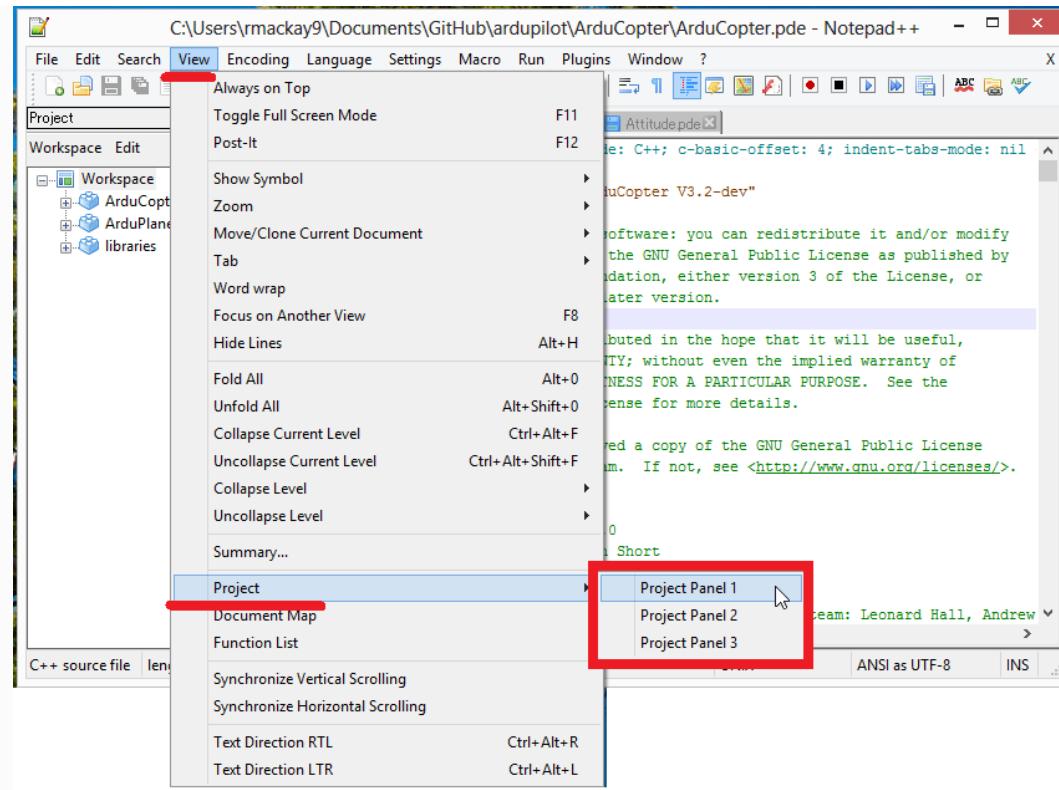
## Code Editing Tools and IDEs

This section contains topics related to code editing tools and IDEs that are commonly used with the ArduPilot project.

### Editing the code with NotePad++

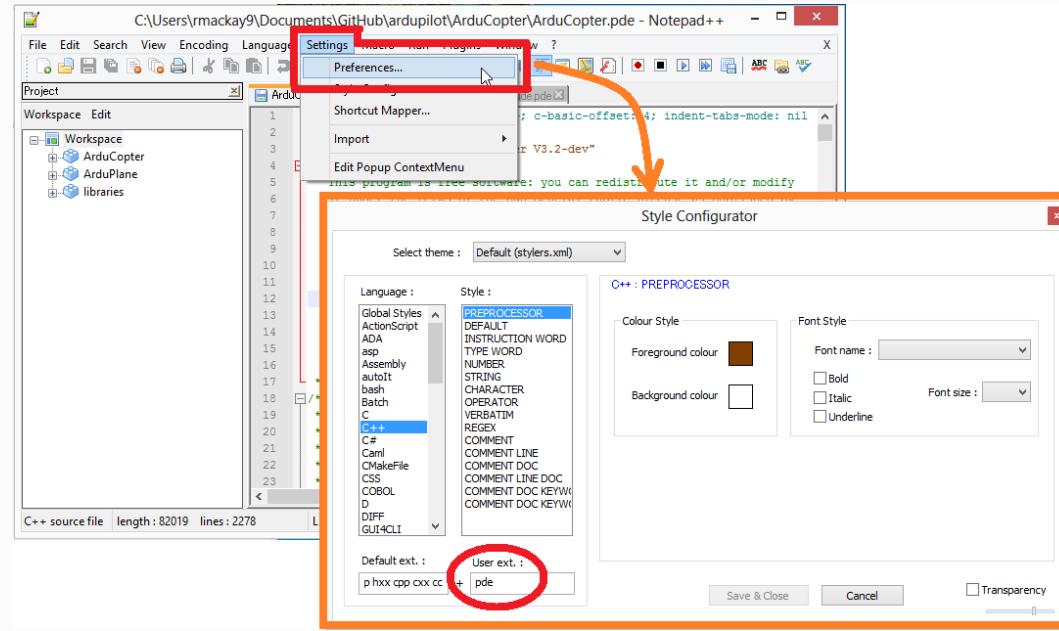
NotePad++ is the most commonly used editor on the development team and can be downloaded from [here](#).

### Use Project Panels to organise lots of files



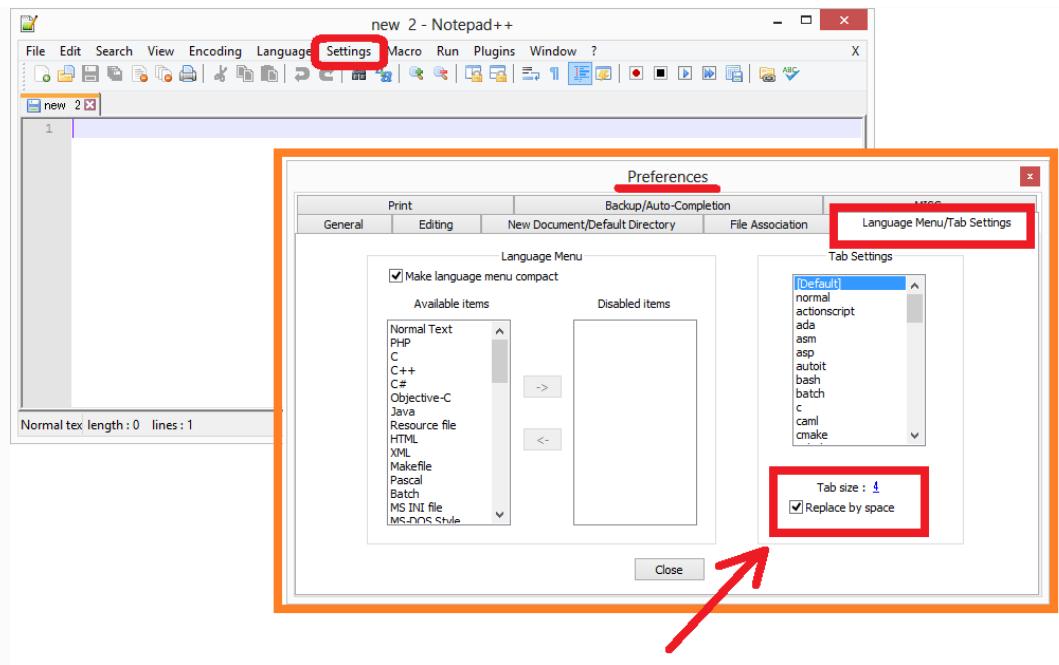
## Format PDE files as C++

Select "Style Configurator" under Preference:



## Use spaces instead of tabs

Under Settings, Preference, Language Menu/Tab Settings ensure the "Replace [tab] by space" checkbox is checked.



## Eclipse IDE

Eclipse is the recommended editor for ArduPilot development on Windows.

The easiest way to set up Eclipse for ArduPilot development is to use the version (Eclipse “Luna”) included with the PX4 Toolchain (this version can be used for both Pixhawk/PX4 and APM development). Instructions for the different controller boards are given below:

- [Building ArduPilot for APM2.x with Eclipse on Windows](#)
- [Building for Pixhawk/PX4 using Eclipse on Windows](#)

## Simulation

This section contains topics about ArduPilot code-testing using simulation. Simulation allows for safe testing of experimental code and settings. Crashing software planes is a lot cheaper than crashing real ones!

Two “classes” of simulation are provided (not all solutions are available for all vehicles)

- **Hardware In the Loop (HITL)** simulation replaces the plane and the environment with a simulator. The simulator has a high-fidelity aircraft dynamics model and environment model (wind, turbulence, etc.). The physical APM hardware is configured exactly as for flight, and connects to your computer running the simulator, rather than the aircraft.
- **Software In The Loop (SITL)** simulation (additionally) virtualizes the autopilot hardware as well as the aircraft and the environment as in HITL. SITL is useful for rapid development and when physical hardware (autopilots and ground stations) are not available or not required.

### Tip

We recommend [Software In The Loop \(SITL\)](#) as the setup is generally easier, it doesn't require vehicle hardware, and it supports all our main vehicle types.

In addition, you can also use:

- **Log replay:** You can [replay binary dataflash log files](#) using the “Replay” system to examine the behaviour of some aspects of the ardupilot internals

## SITL Simulator (Software in the Loop)

The SITL (software in the loop) simulator allows you to run Plane, Copter or Rover without any hardware. It is a build of the autopilot code using an ordinary C++ compiler, giving you a native executable that allows you to test the behaviour of the code without hardware.

This article provides an overview of SITL's benefits and architecture.

### Overview

SITL allows you to run ArduPilot on your PC directly, without any special hardware. It takes advantage of the fact that ArduPilot is a portable autopilot that can run on a very wide variety of platforms. Your PC is just another platform that ArduPilot can be built and run on.

When running in SITL the sensor data comes from a flight dynamics model in a flight simulator. ArduPilot has a wide range of vehicle simulators built in, and can interface to several external simulators. This allows ArduPilot to be tested on a very wide variety of vehicle types. For example, SITL can simulate:

- multi-rotor aircraft
- fixed wing aircraft
- ground vehicles
- camera gimbals
- antenna trackers
- a wide variety of optional sensors, such as Lidars and optical flow sensors

Adding new simulated vehicle types or sensor types is straightforward.

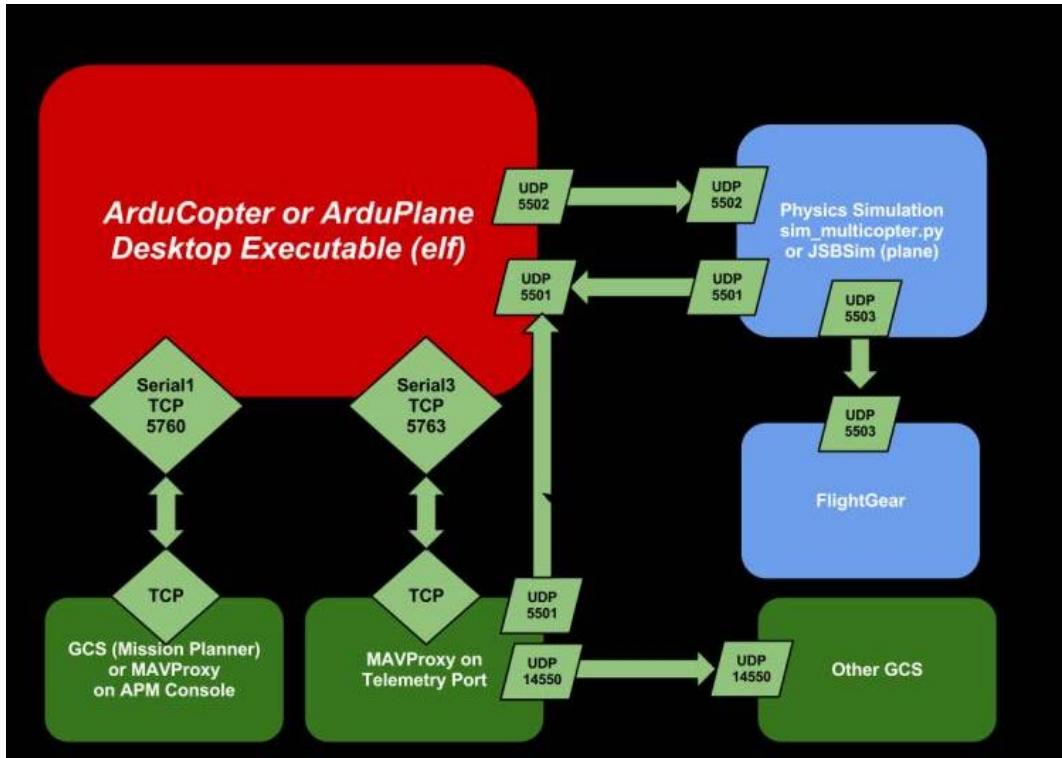
A big advantage of ArduPilot on SITL is it gives you access to the full range of development tools available to desktop C++ development, such as interactive debuggers, static analyzers and dynamic analysis tools. This makes developing and testing new features in ArduPilot much simpler.

### Running SITL

The APM SITL environment has been developed to run natively on both Linux and Windows. For instructions see [Setting up SITL on Linux](#) and [Setting up SITL on Windows](#) for more information.

### SITL Architecture

Note in the image below the port numbers are indicative only and can vary. For instance the ports between ArduPilot and the simulator on the image are 5501/5502 but they can vary to be 5504/5505 or other port numbers depending on your environment.



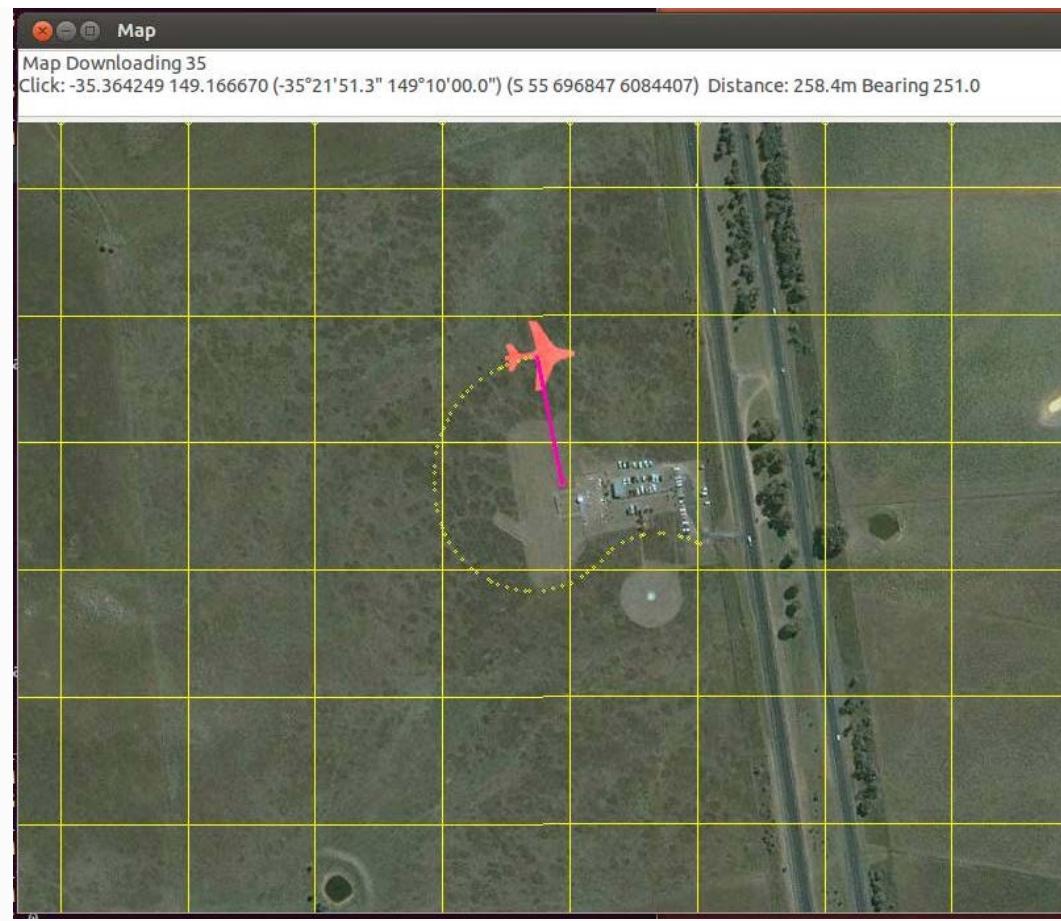
## Setting up SITL on Linux

This page describes how to setup the [SITL \(Software In The Loop\)](#) on Linux. The specific commands were tested on Ubuntu 12.10, 13.04 and 14.10.

### Overview

The SITL simulator allows you to run Plane, Copter or Rover without any hardware. It is a build of the autopilot code using an ordinary C++ compiler, giving you a native executable that allows you to test the behaviour of the code without hardware.

SITL runs natively on Linux and Windows. See the separate [windows installation page](#) for a windows install.



### Install steps [¶](#)

Please follow each of the steps described below.

There is also a linked video below showing how to do the setup.

### Download ardupilot [¶](#)

If you don't have a copy of the ardupilot git repository then open a terminal and run:

```
git clone git://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

### JSBSim (Plane only) [¶](#)

If you want to fly the fixed wing (Plane) simulator then you will need to use the JSBSim flight simulator.

JSBSim is a sophisticated flight simulator that is used as the core flight dynamics system for several well known flight simulation systems.

In the same directory (your home directory) run these commands:

```
git clone git://github.com/tridge/jsbsim.git
sudo apt-get install libtool automake autoconf libexpat1-dev
```

If you are getting an error message saying you need a newer version of JSBSim then you can update it like this:

```
cd jsbsim
git pull
./autogen.sh --enable-libraries
make
```

## Install some required packages¶

If you are on a debian based system (such as Ubuntu or Mint) then run this:

```
sudo apt-get install python-matplotlib python-serial python-wxgtk2.8 python-wxtools python-lxml
sudo apt-get install python-scipy python-opencv ccache gawk git python-pip python-pexpect
sudo pip install future pymavlink MAVProxy
```

Or if you are on a RPM based system (such as Fedora) run this:

```
yum install opencv-python wxPython python-pip pyserial scipy python-lxml python-matplotlib python-pexpect
python-matplotlib-wx
```

## Add some directories to your search path¶

Add the following lines to the end of your ".bashrc" in your home directory (notice the . on the start of that filename. Also, this is a hidden file, so if you're using a file manager, make sure to turn on "show hidden files"). Note the order, it is important to have jsbsim/src before autotest in case you're running a virtual machine.

```
export PATH=$PATH:$HOME/jsbsim/src
export PATH=$PATH:$HOME/ardupilot/Tools/autotest
export PATH=/usr/lib/ccache:$PATH
```

Then reload your PATH by using the “dot” command in a terminal

```
. ~/.bashrc
```

## Start SITL simulator¶

To start the simulator first change directory to the vehicle directory. For example, for the fixed-wing code change to **ardupilot/ArduPlane**:

```
cd ardupilot/ArduPlane
```

Then start the simulator using **sim\_vehicle.py**. The first time you run it you should use the -w option to wipe the virtual EEPROM and load the right default parameters for your vehicle.

```
sim_vehicle.py -w
```

After the default parameters are loaded you can start the simulator normally. First kill the sim\_vehicle.py you are running using Ctrl-C. Then:

```
sim_vehicle.py --console --map --aircraft test
```

### Tip

**sim\_vehicle.py** has many useful options, ranging from setting the simulation speed through to choosing the initial vehicle location. These can be listed by calling it with the **-h** flag (and some are demonstrated in [Using SITL for ArduPilot Testing](#)).

### Load a mission¶

Let's also load a test mission. From within MAVProxy type:

```
wp load ../Tools/autotest/ArduPlane-Missions/CMAC-toff-loop.txt
```

CMAC-toff-loop.txt contains a mission which flies in a loop around my local flying field. Now let's takeoff!

Run the command "arm throttle" followed by "mode auto"

```
arm throttle  
mode auto
```

Your virtual aircraft should now takeoff.

#### Learn MAVProxy¶

To get the most out of SITL you really need to learn to use MAVProxy. Have a read of the [MAVProxy documentation](#). Enjoy flying!

#### Updating MAVProxy and pymavlink¶

New versions of MAVProxy and pymavlink are released quite regularly. If you are a regular SITL user you should update every now and again using this command

```
sudo pip install --upgrade pymavlink MAVProxy
```

#### Using a different JSBSim model¶

If using the JSBSim plane simulator you can specify a different JSBSim model than the default Rascal110 by specifying the model name using the -f parameter to sim\_vehicle.py, like this:

```
sim_vehicle.py -f jsbsim:MyModel --console --map
```

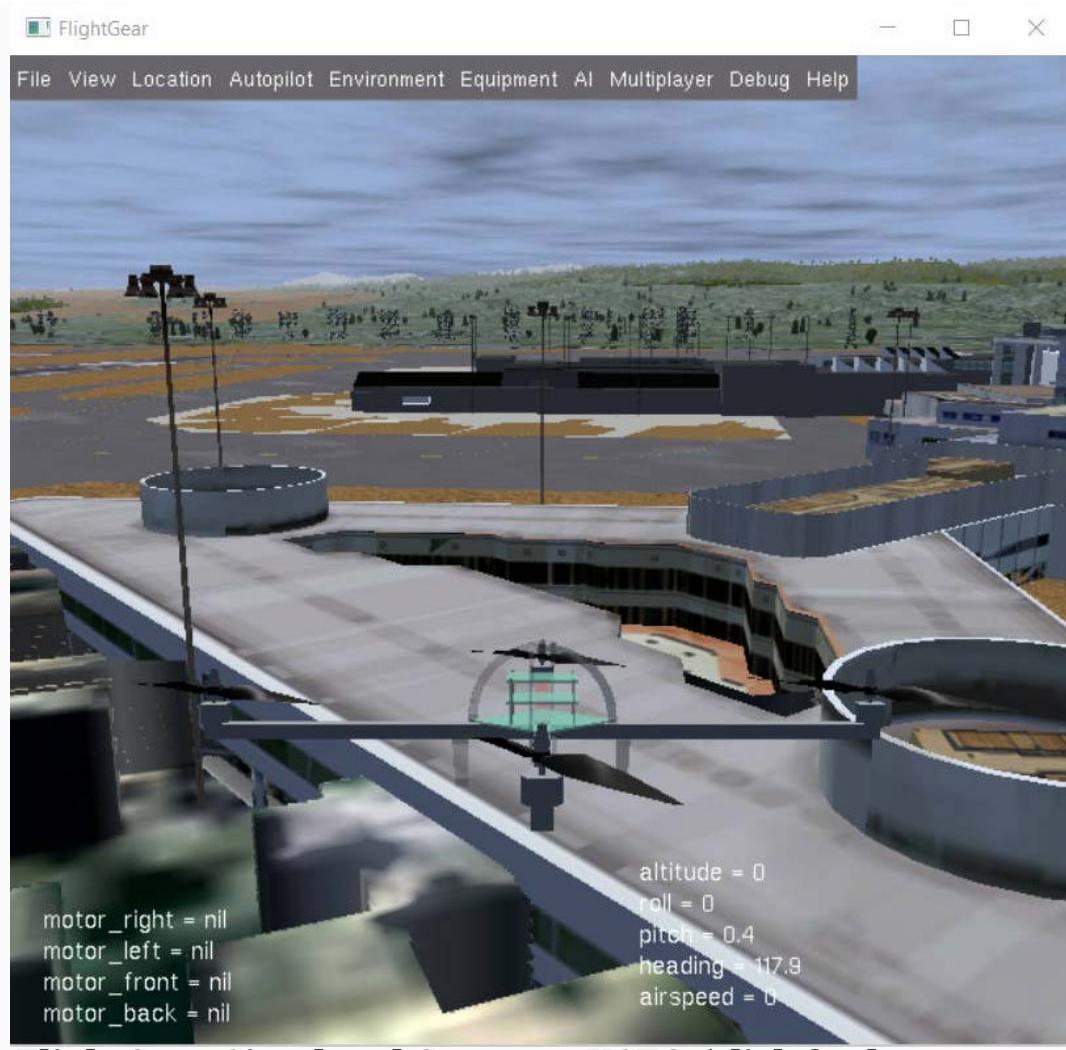
the model should be in the **Tools/autotest/aircraft/** directory.

#### FlightGear 3D View (Optional)¶

Developers can optionally install the [FlightGear Flight Simulator](#) and use it (in view-only mode) to display a 3D simulation of the vehicle and its surroundings. This provides a much better visualization than the 2D maps and HUD flight displays provided by *MAVProxy* and *Mission Planner*.

#### Note

FlightGear support is currently only in master (January 2016). It should appear in the *next* versions of the vehicle codelines (not present on current versions: Copter 3.3, Plane 3.4, Rover 2.5).



### **FlightGear: Simulated Copter at KSFO (click for larger view).**

SITL outputs *FlightGear* compatible state information on UDP port 5503. We highly recommend you start *FlightGear* before starting SITL (although this is not a requirement, it has been found to improve stability in some systems).

The main steps (tested on Ubuntu Linux 14.04 LTS) are:

1. Install *FlightGear* from the terminal:

```
sudo apt-get install flightgear
```

2. Open a new command prompt and run the appropriate shell file for your vehicle in `/ardupilot/Tools/autotest/`: `fg_plane_view.sh` (Plane) and `fg_quad_view.sh` (Copter).

This will start *FlightGear*.

3. Start SITL in the terminal in the normal way. In this case we're specifying the start location as San Francisco airport (KSFO) as this is an interesting airport with lots to see:

```
sim_vehicle.py -j4 -L KSFO
```

#### Note

*FlightGear* will always initially start by loading scenery at KSFO (this is hard-coded into the batch file) but will switch to the scenery for the simulated location once SITL is started.

## Tip

### If the vehicle appear to be hovering in space (no

scenery) then *FlightGear* does not have any scenery files for the selected location. Choose a new location!

You can now takeoff and fly the vehicle as normal for [Copter](#) or [Plane](#), observing the vehicle movement including pitch, yaw and roll.

### Next steps

After installation, see [Using SITL for ArduPilot Testing](#) for guidance on flying and testing with SITL.

## Setting up SITL on Windows

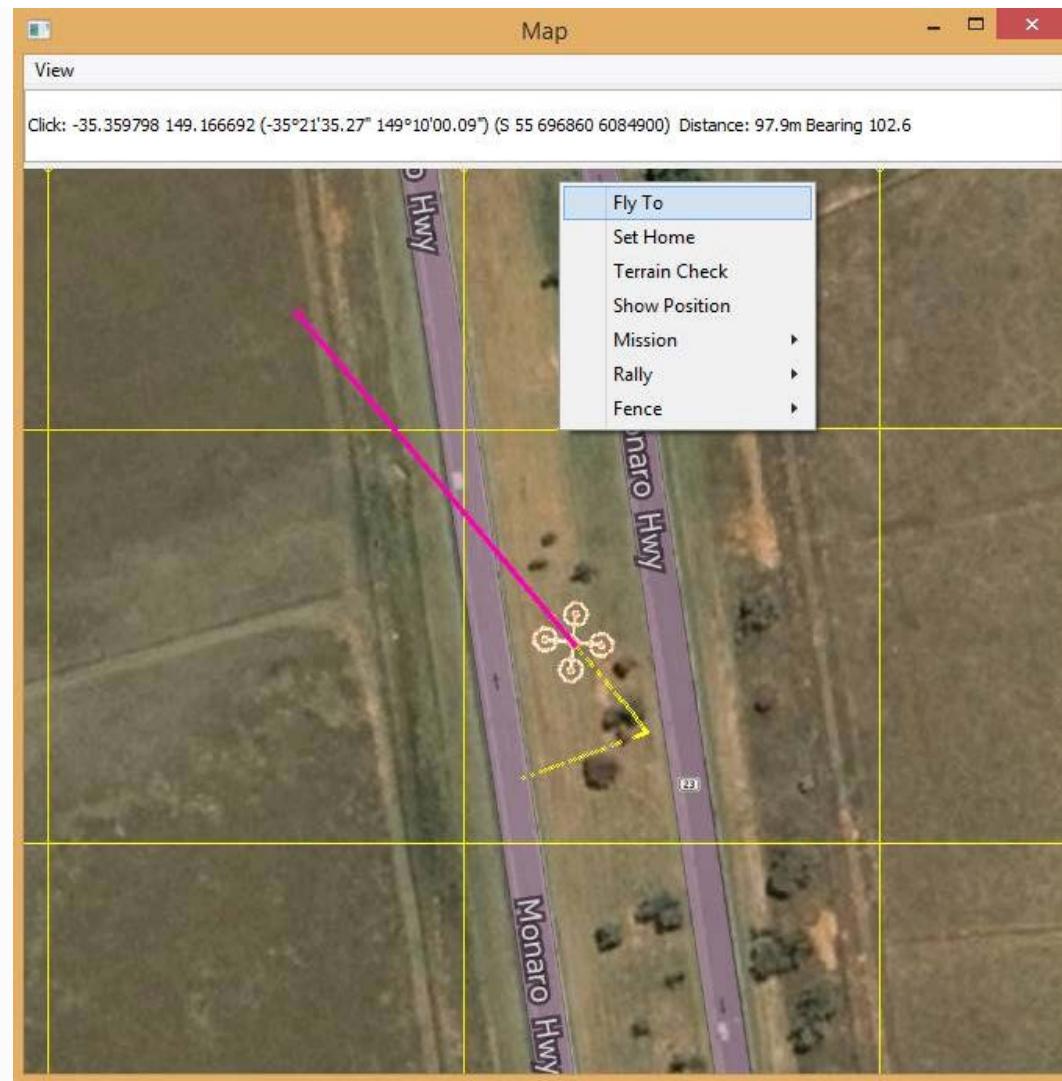
This article shows how to build and run SITL *natively* on Windows. The specific commands were tested on *Windows 8.1* using *Cygwin 2.0.0*, *MAVProxy 1.4.19*, *Mission Planner 1.3.25*, and *AC3.3 (Copter)*.

### Overview

The [SITL \(Software In The Loop\)](#) simulator is a build of the ArduPilot code which allows you to run Plane, Copter or Rover without any hardware.

SITL was originally developed on Linux, and but can now be built and run *natively* on both Linux or Windows. It can also be run on a virtual machine (Linux) hosted on Windows, Mac OSX, or Linux.

These instructions explain how to build SITL *natively* on Windows, and how to interact with the simulator using [MAVProxy](#) and/or [Mission Planner](#).



## ***MAVProxy Map: Guiding a SimulatedCopter***

Installation steps [¶](#)

Install MAVProxy [¶](#)

MAVProxy is a fully-functioning but minimalist console-based GCS that is commonly used for testing and developing ArduPilot:

- [Download MAVProxy for Windows](#) (latest build)
- Install the executable, accepting the license and all other default settings.

Older builds can be obtained from <http://firmware.ardupilot.org/Tools/MAVProxy/>.

Install Cygwin [¶](#)

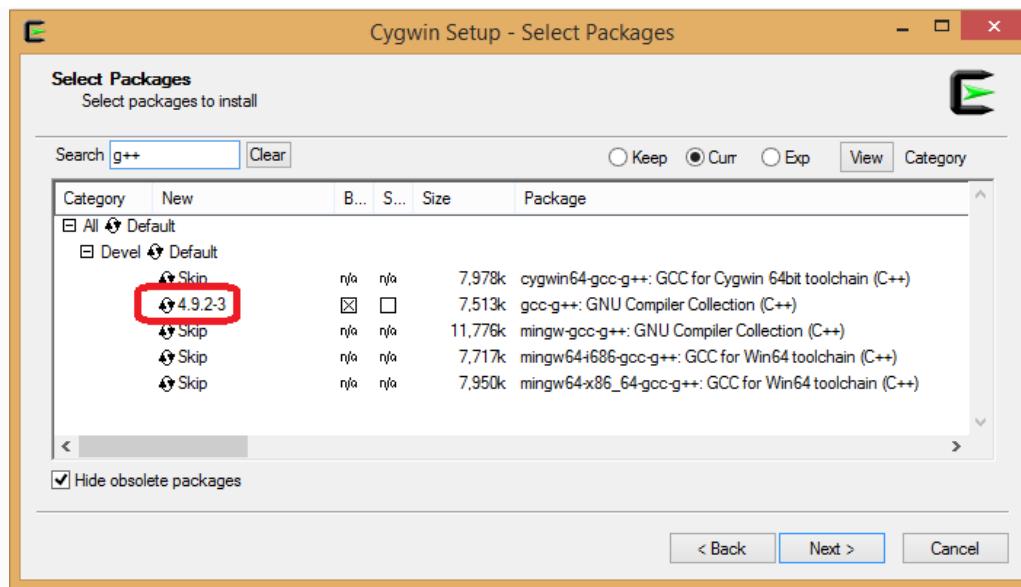
Cygwin provides the tools and libraries that allow us to rebuild ArduPilot on Windows.

1. Download and run the [Cygwin 32-bit installer](#).

Tip

The 32 bit version is preferred over the 64-bit version, which is missing one of our dependencies (procps)

2. Accept the all the prompts (including default file locations) until you reach the *Select Packages* dialog. There are thousands of packages. The easiest way to find the packages is to search on the name. When you've found a needed package click on the **Skip** button to select it for download:



## Cygwin Installer: Select Package Dialog

3. Select the packages listed below (search using the text in the "Name" field):

Name	Category / Name / Description
autoconf	Devel   autoconf: Wrapper scripts for autoconf commands
automake	Devel   automake: Wrapper scripts for automake and aclocal
ccache	Devel   ccache: A C compiler cache for improving recompilation
g++	Devel   gcc-g++ GNU Compiler Collection (C++)
git	Devel   git: Distributed version control system
libtool	Devel   libtool: Generic library support script
make	Devel   make: The GNU version of the 'make' utility
gawk	Interpreters   gawk: GNU awk, a pattern scanning and processing language
libexpat	Libs   libexpat-devel: Expat XML parser library (development files)
libxml2-devel	Libs   libxml2-devel: Gnome XML library (development)
libxslt-devel	Libs   libxslt-devel: XML template library (development files)
python-devel	Python   python-devel: Python language interpreter
procps	System   procps: System and process monitoring utilities (required for pkill)

4. When all the packages are selected, click through the rest of the prompts and accept all other default options (including the additional dependencies).
5. Select **Finish** to start downloading the files.

Set up directories/paths in Cygwin [¶](#)

To save having to set up paths every time you start SITL, it can be helpful to set up the path to the **Tools/autotest** directory.

1. Open and then close the *Cygwin Terminal* from the desktop or start menu icon.

Tip

This will create initialisation files for the user in the Cygwin home directory (and display their locations). For example, a user's home directory might be located at **C:\cygwin\home\user\_name\**.

2. Navigate the file system to the home directory and open the **.bashrc** files (e.g.

**C:\cygwin\home\user\_name.bashrc**.

3. Add the following line to the end of **.bashrc**

```
export PATH=$PATH:$HOME/ardupilot/Tools/autotest
```

The file will be loaded next time you open the *Cygwin terminal*.

#### Tip

Cygwin will not be able to find **sim\_vehicle.py** if you omit this step. This will be reported as a "command not found" error when you try and build:

**sim\_vehicle.py -j4 --map**

Install required Python packages¶

```
python -m ensurepip --user
python -m pip install --user future
python -m pip install --user lxml
```

Download and make ArduPilot¶

Open (reopen) *Cygwin Terminal* and clone the Github [ArduPilot repository](#):

```
git clone git://github.com/ArduPilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

In the terminal navigate to the *ArduCopter* directory and run **make** as shown:

```
cd ~/ardupilot/ArduCopter
make sitl -j4
```

The platform that is built depends on the directory where you run **make** (so this this will build *Copter*).

#### Note

An additional component is required before you can build Plane - see next step!

JSBSim (Plane only)¶

If you want to fly the fixed wing (Plane) simulator then you will need to use the JSBSim flight simulator.

JSBSim is a sophisticated flight simulator that is used as the core flight dynamics system for several well known flight simulation systems. The reason we use JSBSim is that it provides a way to get extremely high frame rate simulation, which is essential for the register level sensor emulation that we use in the SITL build.

Open the *Cygwin Terminal*, navigate to your home directory, and enter:

```
git clone git://github.com/tridge/jsbsim.git
cd jsbsim
./autogen.sh
make
cp src/JSBSim.exe /usr/local/bin
```

Now you can navigate to the ArduPlane directory and build Plane in the same way as described for Copter in the next section ([Running SITL and MAVProxy](#)):

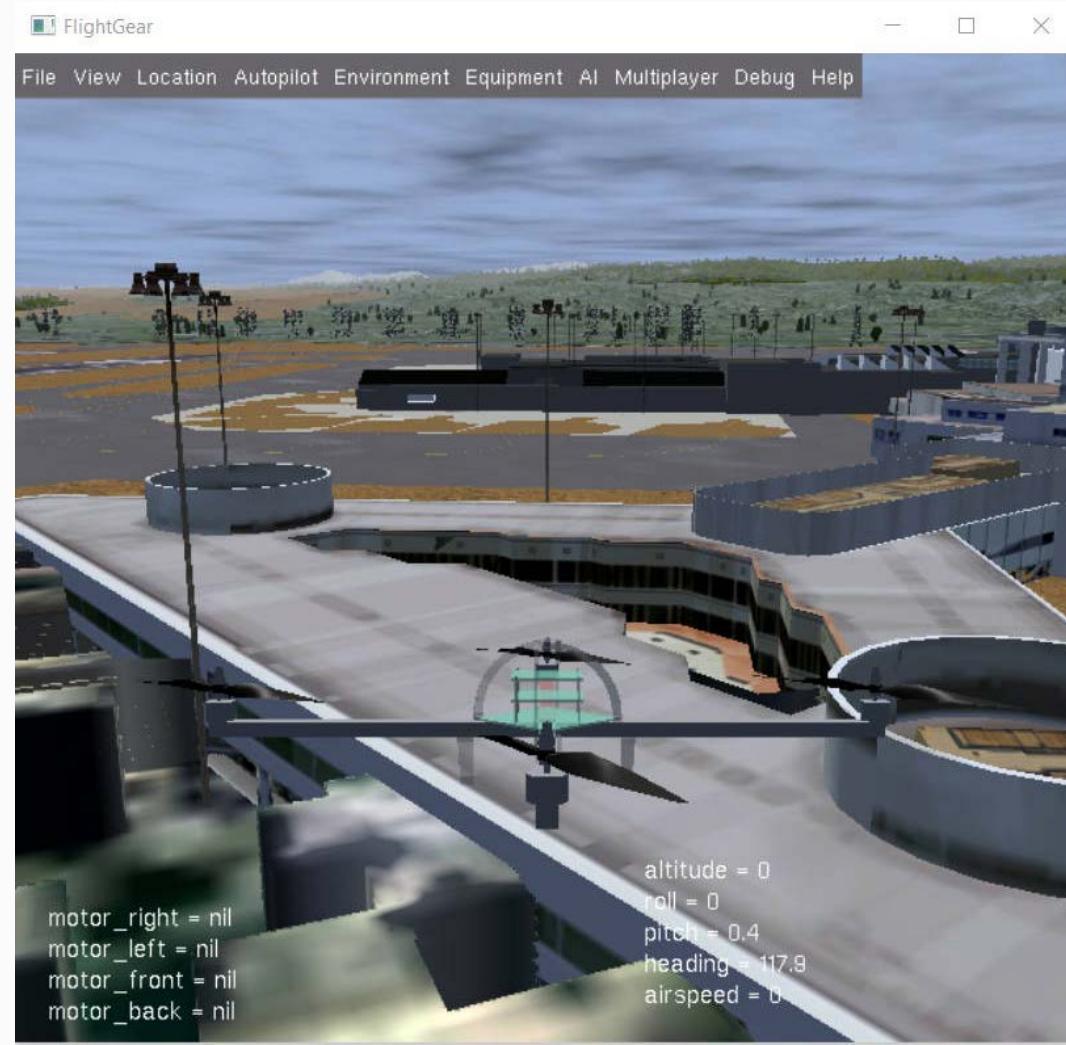
```
cd ~/ardupilot/ArduPlane
sim_vehicle.py -j4 --map
```

### FlightGear 3D View (Optional)

Developers can optionally install the [FlightGear Flight Simulator](#) and use it (in view-only mode) to display a 3D simulation of the vehicle and its surroundings. This provides a much better visualization than the 2D maps and HUD flight displays provided by *MAVProxy* and *Mission Planner*.

#### Note

FlightGear support is currently only in master (January 2016). It should appear in the *next* versions of the vehicle codelines (not present on current versions: Copter 3.3, Plane 3.4, Rover 2.5).



### ***FlightGear: Simulated Copter at KSFO (click for larger view).***

SITL outputs *FlightGear* compatible state information on UDP port 5503. We highly recommend you start *FlightGear* before starting SITL (although this is not a requirement, it has been found to improve stability in some systems).

The main steps are:

1. Download [FlightGear 3.4.0](#)

Warning

**At time**

of writing [version 3.4.0 is required on Windows.](#)

2. Open a new command prompt and run the appropriate batch file for your vehicle in

/ardupilot/Tools/autotest/: [fg\\_plane\\_view.bat](#) (Plane) and [fg\\_quad\\_view.bat](#) (Copter).

This will start FlightGear.

3. Start SITL in Cygwin in the normal way. In this case we're specifying the start location as San Francisco airport (KSFO) as this is an interesting airport with lots to see:

```
sim_vehicle.py -j4 -L KSFO
```

Note

**FlightGear will always initially start by loading scenery at**

KSFO (this is hard-coded into the batch file) but will switch to the scenery for the simulated location once SITL is started.

Tip

**If the vehicle appear to be hovering in space (no**

scenery) then *FlightGear* does not have any scenery files for the selected location. Choose a new location!

You can now takeoff and fly the vehicle as normal for [Copter](#) or [Plane](#), observing the vehicle movement including pitch, yaw and roll.

Troubleshooting

A very small number of users have reported build errors related to Windows not setting paths correctly.

For more information see [this issue](#).

Running SITL and MAVProxy

MAVProxy is commonly used by developers to communicate with SITL. To build and start SITL for a 4-core CPU and then launch a *MAVProxy map*:

1. Navigate to the target vehicle directory (in this case Copter) in the *Cygwin Terminal* and call

[sim\\_vehicle.py](#) to start SITL:

```
cd ~/ardupilot/ArduCopter
sim_vehicle.py -j4 --map
```

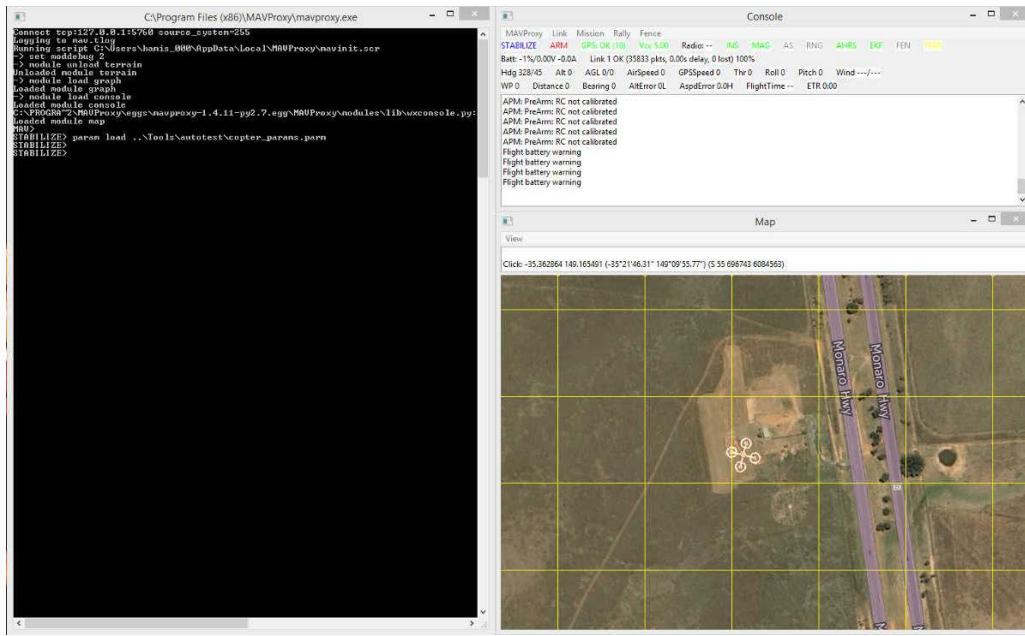
If you get a windows security alert for the the firewall, allow the connection.

Tip

[sim\\_vehicle.py](#) has many useful build options, ranging from setting the simulation speed through to choosing the initial vehicle location. These can be listed by calling it with the [-h](#) flag (and some are demonstrated in [Using SITL for ArduPilot Testing](#)).

2. SITL and MAVProxy will start. MAVProxy displays three windows:

- A command prompt in which you enter commands to SITL
- A Console which displays current status and messages
- A map that shows the current position of the vehicle and can be used (via right-click) to control vehicle movement and missions.

**Tip**

It is useful to arrange the windows as shown above, so you can observe the status and send commands at the same time.

- Configure the vehicle by loading some standard/test parameters into the *MAVProxy command prompt*:

```
param load ..\Tools\autotest\copter_params.parm
```

- You can send commands to SITL from the command prompt and observe the results on the map.

- Change to GUIDED mode, arm the throttle, and then takeoff:

```
mode guided
arm throttle
takeoff 40
```

Watch the altitude increase on the console.

**Note**

Takeoff must start within 15 seconds of arming, or the motors will disarm.

- Change to CIRCLE mode and set the radius to 2000cm

```
mode circle
param set circle_radius 2000
```

Watch the copter circle on the map.

- When you're ready to land you can set the mode to RTL (or LAND):

```
mode rtl
```

This is a very basic example. For links to more information on what you can do with SITL and MAVProxy see the section: [Next Steps](#).

**Adding additional GCS with MAVProxy**

You can attach multiple additional ground control stations to SITL from *MAVProxy*. The simulated vehicle can then be controlled and viewed through any attached GCS.

First use the `output` command on the *MAVProxy command prompt* to determine where *MAVProxy* is sending packets:

```
GUIDED> output
GUIDED> 2 outputs
0: 127.0.0.1:14550
1: 127.0.0.1:14551
```

This tells us that we can connect *Mission Planner* to either UDP port 14550 or 14551, as shown on the dialog below.



### ***Mission Planner: Connecting to a UDPPort***

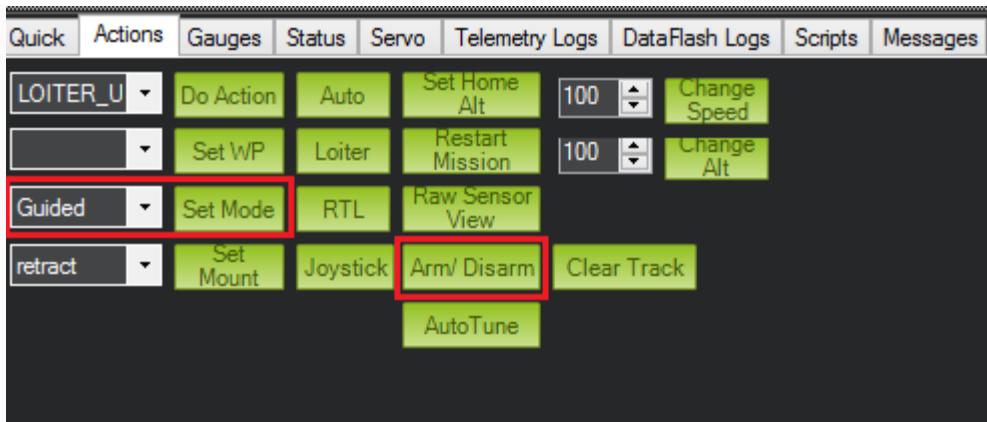
#### Tip

We could connect *APM Planner 2* to the remaining port. If we needed a third port, we could add it as shown:

```
GUIDED> output add 1: 127.0.0.1:14553
```

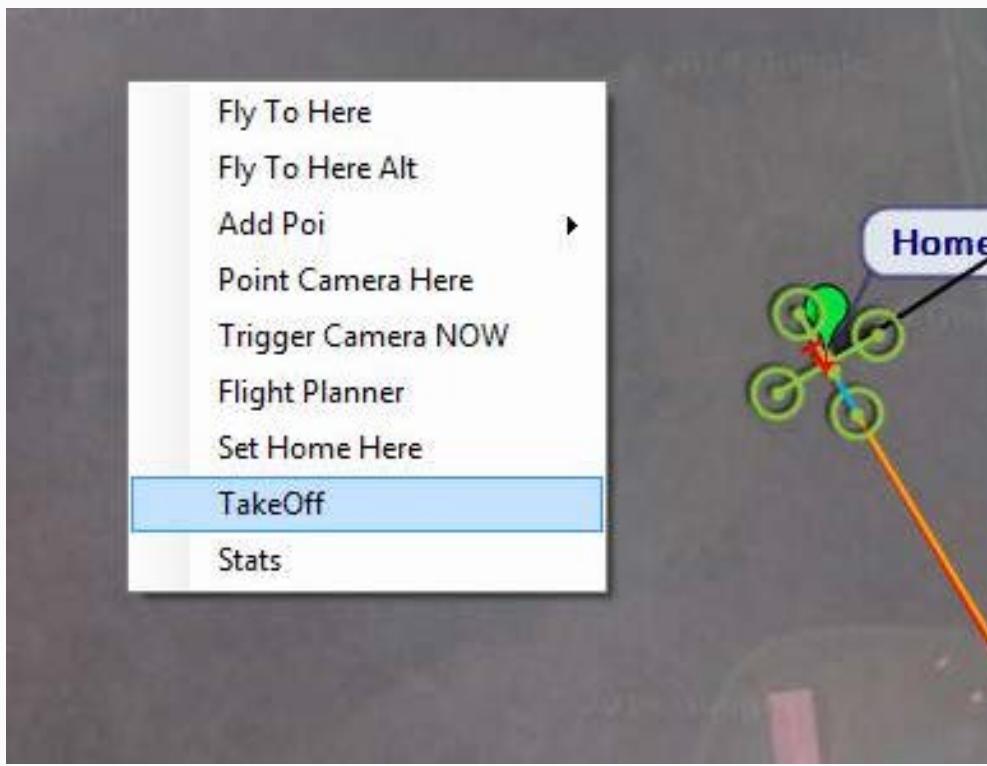
*Mission Planner* can then be used to control the simulated vehicle in exactly the same way as though it were a real vehicle. We can reproduce the previous “takeoff-circle-land” example as shown below:

1. Change to GUIDED mode, arm the throttle, and then takeoff
  - Open the *FLIGHT DATA* screen and select the *Actions* tab on the bottom left. This is where we can change the mode and set commands.



### **Mission Planner: Actions Tab (Set Mode, Arm/Disarm)**

- Select **Guided** in the *Mode selection list* and then press the **Set Mode** button.
- Select the **Arm/Disarm** button
- Right-click on the map and select Takeoff. Then enter the desired takeoff altitude



### **Mission Planner Map: Takeoff Command**

#### Note

Takeoff must start within 15 seconds of arming, or the motors will disarm.

2. Change to CIRCLE mode on the *Action* tab and watch the copter circle on the map.
3. You can change the circle radius in the *CONFIG/TUNING* screen. Select *Full Parameters List*, then the **Find** button and search for **CIRCLE\_MODE**. When you've changed the value press the **Write Params** button to save them to the vehicle.
4. When you're ready to land you can set the mode to RTL.

Running SITL with a GCS without MAVProxy

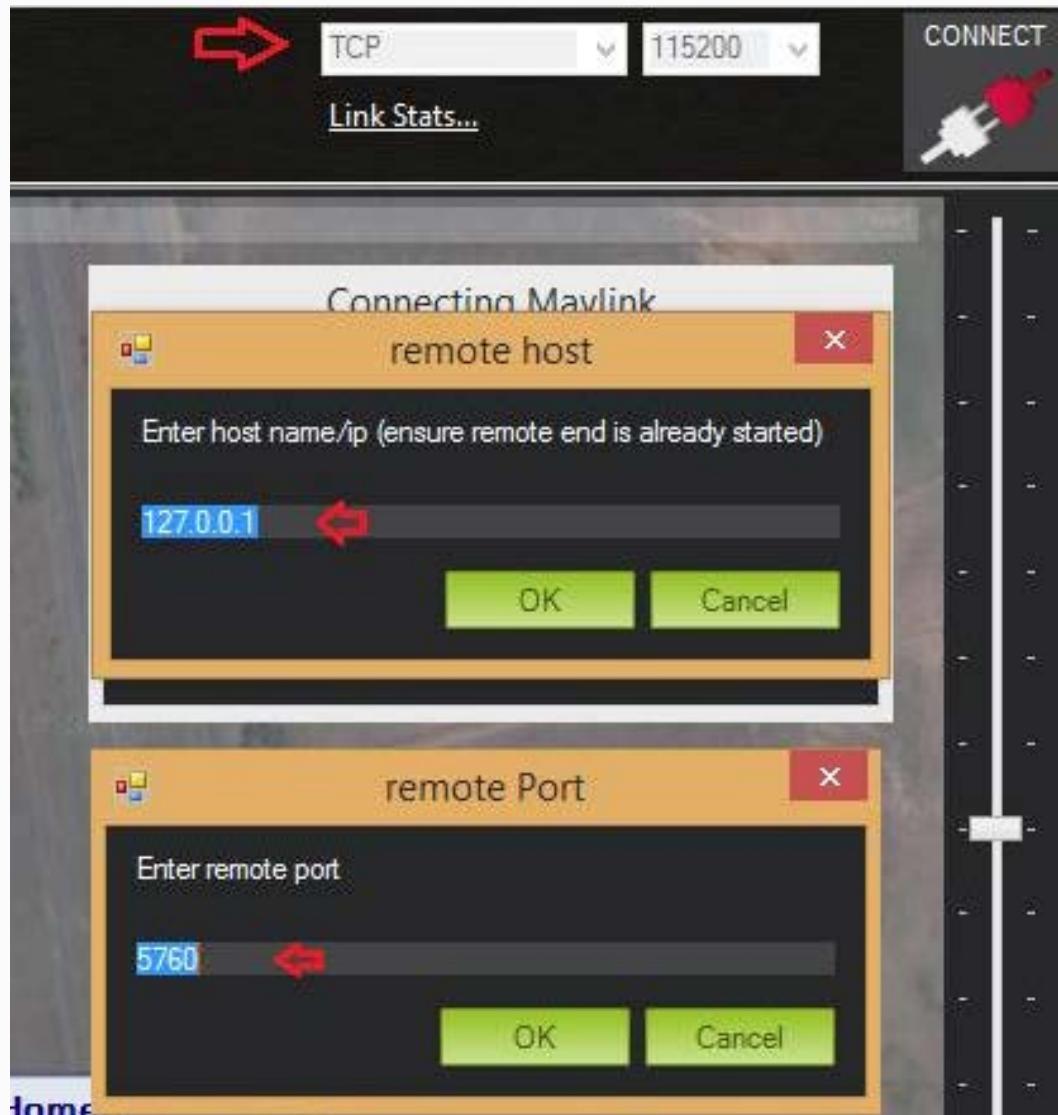
It is also possible to interact with SITL without using MAVProxy at all using **ArduCopter.elf** (in the **ArduCopter** directory).

Run the file in the *Cygwin Terminal*, specifying a home position and vehicle model as shown below:

```
hamis_000@XPS12ultra ~/ardupilot/ArduCopter
$ ./ArduCopter.elf --home -35,149,584,270 --model quad
Started model quad at -35,149,584,270 at speed 1.0
Starting sketch 'ArduCopter'
Starting SITL input
bind port 5760 for 0
Serial port 0 on TCP port 5760
Waiting for connection ....
```

The command output shows that you can connect to SITL using TCP/IP at port 5760.

In *Mission Planner* we first change the link type to TCP and then press the **Connect** button. Click through the *remote host* and *remote Port* prompts as these default to the correct values.



### ***Mission Planner: Connecting toSITL using TCP***

Mission Planner will then connect and can be used just as before.

Tip

**ArduCopter.elf** has other startup options, which you can use using the **-h** command line parameter:

```
./ArduCopter.elf -h
```

## Updating ArduPilot<sup>¶</sup>

The ArduPilot source is cloned to the Windows Cygwin home directory (e.g.

`C:\cygwin\home\user_name\ardupilot`). Developers can edit the source in `ardupilot/`, or update it using

`git pull`.

Similarly, the JSBSim source can be updated by calling `git pull` in the `jsbsim/` directory.

## Updating MAVProxy<sup>¶</sup>

Warning

The *MAVProxy 1.4.19 \*installer does not properly remove all parts of preceding installations. Before installing a new version you must first delete the old directory: C\*:Program Files (x86)MAVProxy\*\*.* Simply [Download and Install MAVProxy for Windows](#) (this link always points to the latest version!)

### Next steps<sup>¶</sup>

SITL and MAVProxy can do a whole lot more than shown here, including manually guiding the vehicle, and creating and running missions. To find out more:

- Read the [MAVProxy documentation](#).
- See [Using SITL for ArduPilot Testing](#) for guidance on flying and testing with SITL.

## Setting up SITL using Vagrant

This article explains how to set up the [SITL ArduPilot Simulator](#) in a virtual machine environment using [Vagrant](#), and connect it to a Ground Control Station running on the host computer. This approach is much easier and faster than [manually](#) setting up a virtual machine to run SITL on Mac OSX or Windows (or Linux).

These instructions have been tested on Windows 8.1.

### Overview<sup>¶</sup>

The SITL (Software In The Loop) simulator allows you to run Plane, Copter or Rover without any hardware. The simulator runs the normal ArduPilot code as a native executable on a Linux PC. SITL can also be run within a virtual machine on Windows, Mac OSX or Linux.

Vagrant is a tool for automating setting up and configuring development environments running in virtual machines. While it is possible to [manually set up SITL to run in a VM on Windows](#) (or Mac OSX), it is **much easier** (and more reproducible) to use Vagrant to do this work for you.

### Note

Due to the way submodules are currently handled in the build system, it is not possible to have a repository which can be built on both the host and virtual machines. A dedicated repository should be used for running the Vagrant virtual machine.

### Preconditions<sup>¶</sup>

- **Git (1.8.x or later)** must be installed on the host computer.
  - [Git for Windows](#) (1.9.5) is recommended.

### Note

**The current windows px4 toolchain (v14) does not have a recent enough version of GIT**

### Warning

**You must use the newer**

version for the git submodule init step. After that step you can use an older version.

- Ensure that `git` is set to leave line endings untouched. Click on your new “Git Shell (or Bash)” Icon (the terminal was installed when you installed `git`) and type in the following in the Git “MINGW32” Terminal window:

```
git config --global core.autocrlf false
```

- SSH must be installed on the host computer and be added to the system PATH. SSH is installed with GIT, or you can install it independently for your platform.

### Building PX4 Firmware - Rsync

If you’re using this Vagrant file for the purpose of building PX4 firmware you will probably also wish to install `Rsync`, as this significantly speeds up PX4 builds. The relative build times using different approaches (on a 3Ghz i5 Haswell / 16 Gb) are shown below:

- Native windows px4 toolchain (gcc/mingw): 190 minutes
- Vagrant with shared folders (the default setup described here): 15 minutes 30s (12x faster)
- Vagrant with rsync folders: 1 minute 40s (120x faster)!.

#### Tip

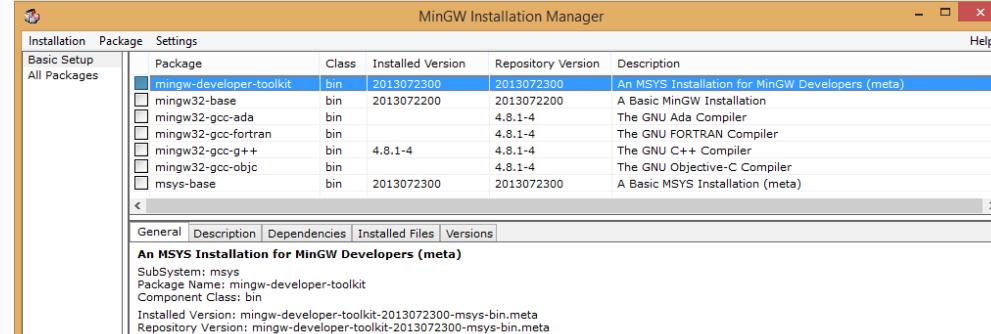
Everything is set up to use shared folders by default. The only caveat is that `px4-clean` does not work with shared folders. You either need to use rsync or run `px4-clean` from outside Vagrant (you can also get usable results by cd’ing to each module subdirectory and running ```git clean -x -d -f`` but don’t do this from the top level otherwise you will delete your vagrant temporary files.`

If you want to use rsync you need to:

- Uncomment the appropriate line in the Vagrantfile:

```
# config.vm.synced_folder ".", "/vagrant", type: "rsync", rsync_auto: true
```

- `rsync` must be installed on the host computer and be added to the system PATH. According to [the vagrant rsync guide](#) you can install rsync from either [Mingw](#) or [Cygwin](#). Assuming you’re using Mingw:
  - [Download the latest Mingw installer](#)
  - In the installer, select and install the `mingw-developer-toolkit`



### MinGW Installation Manager

- Use `mingw-get` to install rsync (you may need to add `mingw-get` to your path):

```
mingw-get install msys-rsync
```

- Set the system path to point to rsync (By default this is in `C:\MinGW\msys\1.0\bin`.)

### Set up the Vagrant and the virtual machine

1. [Download and install VirtualBox.](#)
2. [Download and install Vagrant for your platform. Windows, OS-X and Linux are supported.](#)
3. Clone the [ArduPilot Github repository anywhere on your PC:](#)

```
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
```

4. Start a vagrant instance

- Open a command prompt and navigate to any directory in the [/ArduPilot/ardupilot/Tools/vagrant/](#) source tree.
- Run the command:

```
vagrant up
```

This starts running a VM, based on a *Vagrant configuration file* in the source tree. All the files in this directory tree will “magically” appear inside the running instance at `/vagrant`.

#### Note

The first time you run the `vagrant up` command it will take some time complete. The command needs to fetch a Vagrant base VM and configure it with the development environment.

5. Initialise git submodules

- The ArduPilot source tree references other repositories as *submodules*. These must be initialised by working on the virtual machine:

```
vagrant ssh
git submodule update --init --recursive
exit
```

#### Start running SITL

Enter the following in your vagrant shell to run the Copter simulator. This will first build the code (if it has not previously been built) and then run the simulator:

```
vagrant ssh -c "sim_vehicle.py -j 2"
```

Once the simulation is running, you will start getting information from the MAVLink prompt about vehicle state. For example:

```
GPS lock at 0 meters
APM: PreArm: RC not calibrated
APM: Copter V3.3-dev (999710d0)
APM: Frame: QUAD
APM: PreArm: RC not calibrated
```

The Copter Simulator is built by default, but you can instead build for the plane or rover using the `-v` option:

```
vagrant ssh -c "sim_vehicle.py -j 2 -v ArduPlane"
vagrant ssh -c "sim_vehicle.py -j 2 -v APMrover2"
```

#### Tip

[sim\\_vehicle.py](#) has many useful build options, ranging from setting the simulation speed through to

choosing the initial vehicle location. These can be listed by calling it with the `-h` flag (and some are demonstrated in [Using SITL for ArduPilot Testing](#)).

Run Mission Planner or MAVProxy in your main OS¶

You can now connect to the running simulator from your main OS. Just connect to UDP port 14550, either from *Mission Planner* or *MAVProxy*. The *MAVProxy* command is:

```
mavproxy.py --master=127.0.0.1:14550
```

Shutting down the simulator¶

When you are done with the simulator:

- Press **ctrl-d** in the Vagrant SSH window to exit the special *MAVProxy* that is gluing everything together.
- Suspend the running VM by entering the following in the command prompt:

```
vagrant suspend
```

Restarting the simulator¶

When you need the simulator again you can resume the VM and restart the simulator as shown:

```
vagrant up
vagrant ssh -c "sim_vehicle.py -j 2"
```

Note

Restarting the environment usually only takes a few seconds as the VM is only suspended and the simulation code for the vehicle has already been built.

Updating the simulator¶

The simulator is built from the source tree shared between the host and virtual machines, and any changes will trigger a rebuild next time you start the simulator. To update the simulator you simply need to modify the source tree (or pull a new version from Github).

Next steps¶

To get the most out of SITL we recommend you [Learn MavProxy](#).

The topic [Using SITL for ArduPilot Testing](#) explains how to use the simulator, and covers topics like how to use SITL with Ground Stations other than Mission Planner and MAVProxy.

## Copter SITL/MAVProxy Tutorial

This tutorial provides a basic walk-through of how to use [SITL](#) and [MAVProxy](#) for *Copter* testing.

Overview¶

The article is intended primarily for developers who want to test new Copter builds and bug fixes using SITL and MAVProxy. It shows how to take off, fly in [GUIDED](#) mode, run missions, set a geofence, and perform a number of other basic testing tasks.

The tutorial is complementary to the topic [Using SITL for ArduPilot Testing](#).

Note

- We use *MAVProxy* here, but you can [attach another ground station to SITL](#) if you prefer (similar instructions can be used in any GCS).
- This tutorial is for Copter - see [Plane](#) and [Rover](#) for similar tutorials on the other vehicles.

## Preconditions

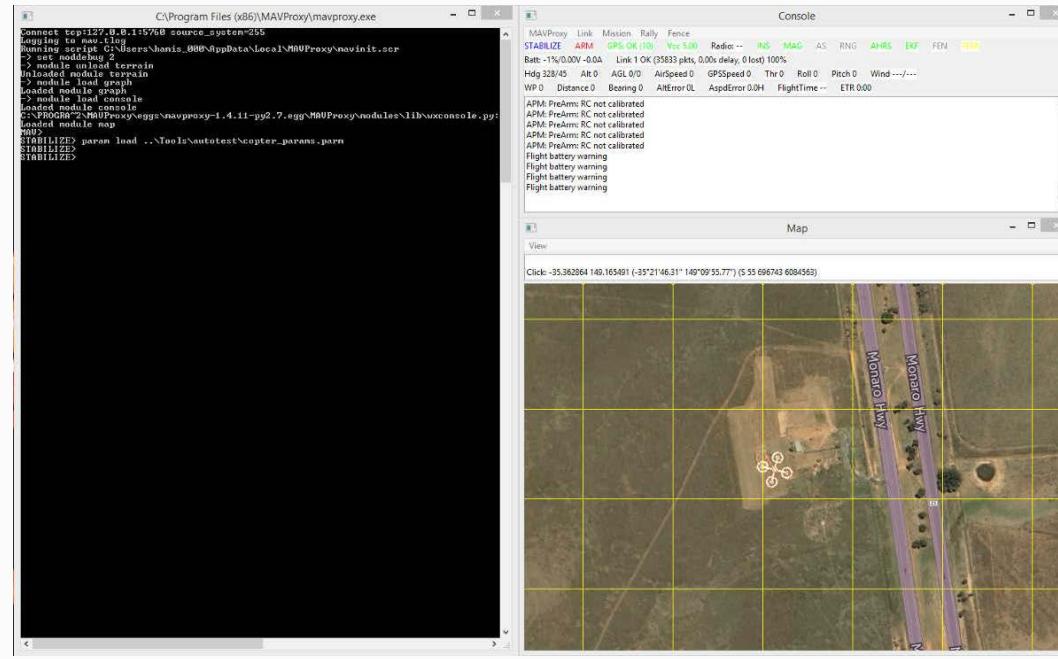
The tutorial assumes you have already set up SITL on [Windows](#) or [Linux](#) and that you have started SITL using the `--map` and `--console` options:

```
cd ~/ardupilot/ArduCopter
sim_vehicle.py -j4 --map --console
```

As part of the setup you should have loaded some standard/test parameters into the *MAVProxy Command Prompt*:

```
param load ..\Tools\autotest\copter_params.parm
```

The *MAVProxy Command Prompt*, *Console* and *Map* should be arranged conveniently so you can observe the status and send commands at the same time.



## Taking off

This section explains how to take off in [GUIDED](#) mode. The main steps are to change to [GUIDED](#) mode, arm the throttle, and then call the `takeoff` command. Takeoff must start within 15 seconds of arming, or the motors will disarm!

### Note

At time of writing, Copter only supports takeoff in guided mode; if you want to fly a mission you first have to take off and then switch to [AUTO](#) mode. From AC3.3 it will be possible to take off in AUTO mode too.

Enter the following commands in the *MAVProxy Command Prompt*.

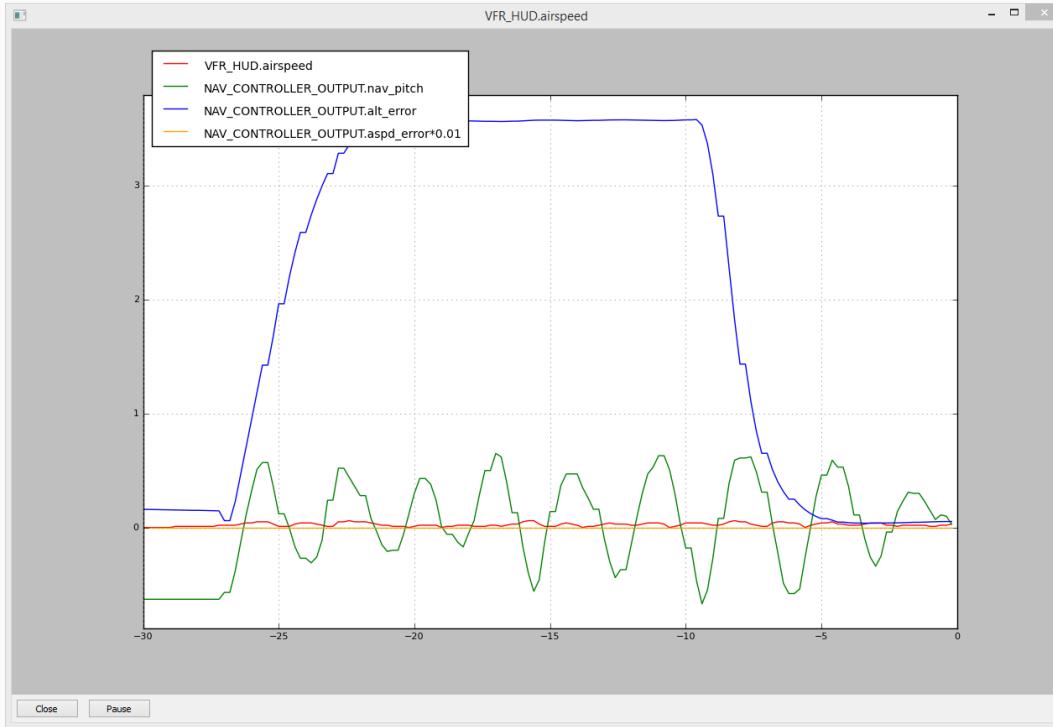
```
mode guided
arm throttle
takeoff 40
```

Copter should take off to an altitude of 40 metres and then hover (while it waits for the next command).

## Monitoring

During takeoff you can watch the altitude increase on the console in the *A/t* field.

Developers may find it useful to **graph** the takeoff by first entering the `gtakeoff` command.



## MAVProxy:Copter Takeoff Graph (gtakeoff)

### Troubleshooting

The most common sources of difficulty taking off are:

1. Starting in any mode other than `GUIDED`.
2. Attempting to takeoff when the vehicle is not armed. This can happen if you call `takeoff` too slowly after `arm throttle`, or if the vehicle fails pre-arm checks.

You can list all *enabled* checks using the command `arm list`:

```
LAND> arm list
LAND> all
params
voltage
compass
battery
ins
rc
baro
gps
```

You can enable and disable checks using `arm check n` and `arm uncheck n` respectively, where n is the name of the check. Use `n` value of `all` to enables/disable all checks.

### Changing flight mode - circle and land

The command below shows how to put Copter into `CIRCLE` mode with a `CIRCLE_RADIUS` of 2000cm. This will fly the Copter in a circle at a constant altitude, with the front pointed towards the centre of the circle.

```
mode circle
param set circle_radius 2000
```

### Note

If you set the `CIRCLE_RADIUS` to zero the vehicle will rotate in place.

Copter supports a [number of other flight modes](#), which you can list in MAVProxy using the `mode` command:

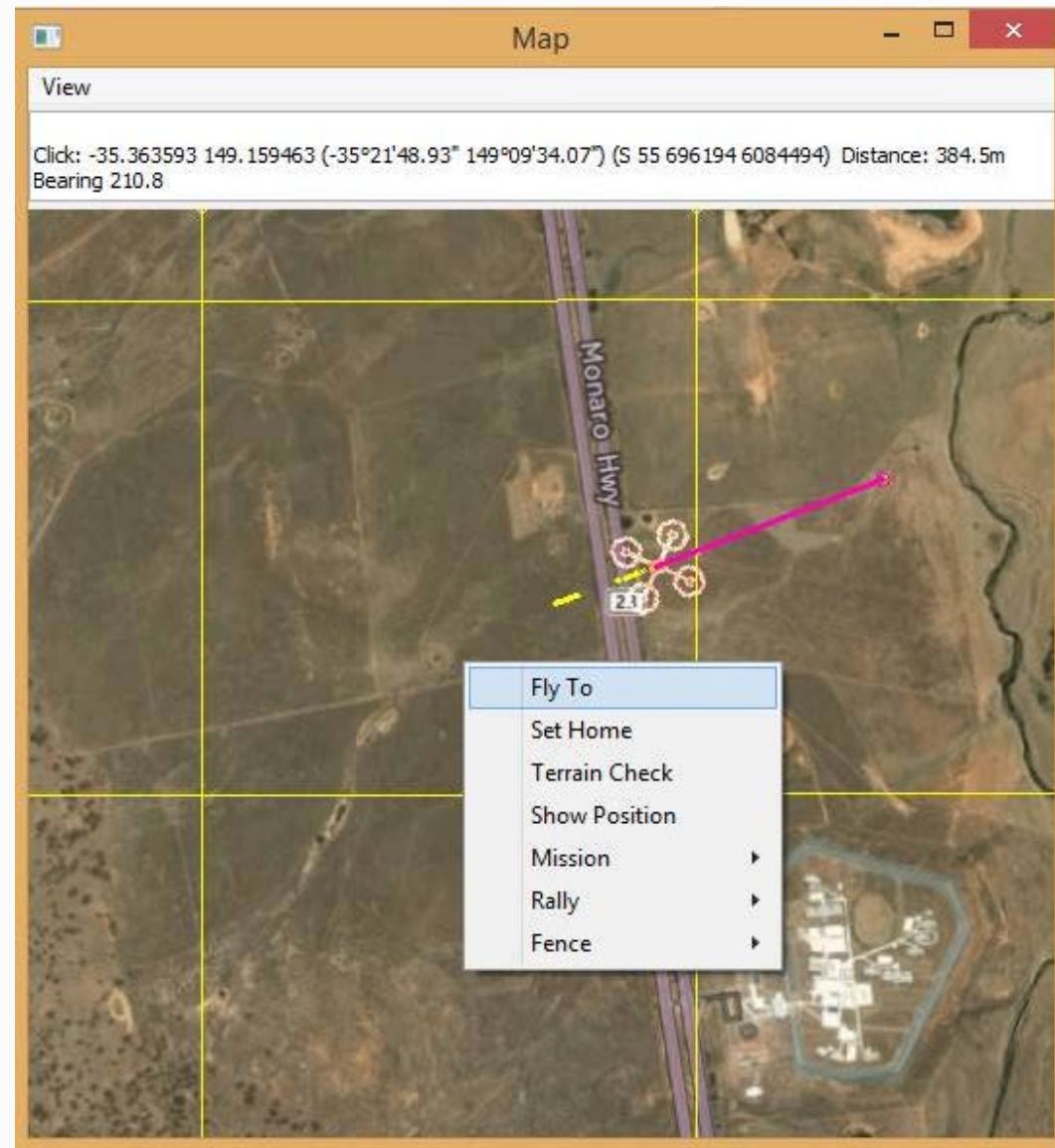
```
LAND> mode
LAND> ('Available modes: ', ['RTL', 'POSHOLD', 'LAND', 'OF_LOITER', 'STABILIZE', 'AUTO', 'GUIDED', 'DRIFT',
'FLIP', 'AUTOTUNE', 'ALT_HOLD', 'LOITER', 'POSITION', 'CIRCLE', 'SPORT', 'ACRO'])
```

As shown above, you can change the mode by specifying `mode modename`. Many of the modes can be set by just entering the mode name, e.g. `rtl`, `auto`, `stabilize` etc.

For example, to land right where you are you would use the command: `mode land`. To return to the launch point and then land you would use the command: `rtl`.

#### Guiding the vehicle

Once you've taken off you can move the vehicle around the map in `GUIDED` mode. The easiest way to do this is to right-click on the map where you want to go, select **Fly to**, and then enter the target altitude.



### **MAVProxy: Fly toLocation**

You can also enter the target position manually on the command line using the two formats below. If only the altitude is specified, the last specified LAT/LON will be used.

```
guided ALTITUDE
guided LAT LON ALTITUDE
```

In addition to `takeoff`, you can send the following commands in `GUIDED` mode:

```
yaw ANGLE ANGULAR_SPEED MODE (MODE is 0 for "absolute" or 1 for "relative")
speed SPEED_VALUE
velocity x y z (m/s)
```

#### Note

These commands correspond to [MAV\\_CMD\\_NAV\\_TAKEOFF](#), [MAV\\_CMD\\_DO\\_CHANGE\\_SPEED](#), [MAV\\_CMD\\_CONDITION\\_YAW](#), [SET\\_POSITION\\_TARGET\\_LOCAL\\_NED](#).

At time of writing, the other [Copter Commands](#) are not supported ([MAVProxy #150](#))

Flying a mission ¶

You can load a mission at any time using the `wp load` command. After you've taken off the current mission will start as soon as you change to `AUTO` mode.

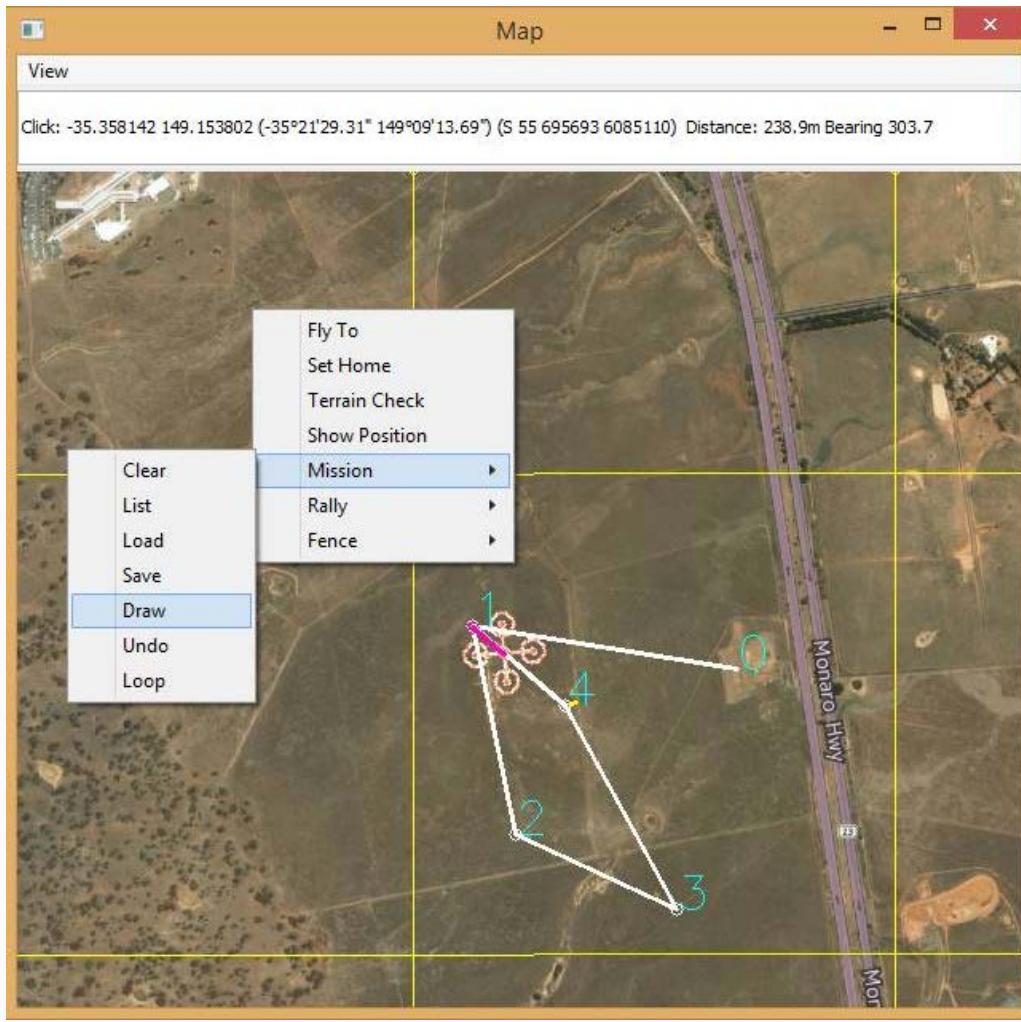
The example below shows how to load and start one of the test missions, skip to the second waypoint, and *loop* the mission:

```
wp load ..\Tools\autotest\CMAC-circuit.txt
mode auto
wp set 2
wp loop
```

The [MAVProxy Waypoints documentation](#) lists the full set of available commands (or you can get them using auto-completion by typing “wp” on the command line).

If you want to create a waypoint mission, this is most easily done on the map:

1. Right-click on the map and then select **Mission | Draw**.



## MAVProxy: Draw Mission Menu

2. Left-click on the map where you want the points to appear.

### Note

Nothing visible will happen when you make the first click. After the second click, lines will join your points to show the path.

3. When you're done, you can loop the mission by right-clicking on the map and selecting **Mission | Loop**.

This approach only allows you to create `MAV_CMD_NAV_WAYPOINT` commands. You can edit missions and use other commands on Linux using the `misseditor` module ([module load misseditor](#)). This is currently broken on Windows. It is also possible to load other types of commands from files.

### Setting a GeoFence

A GeoFence is a virtual barrier that Copter uses to constrain the movement of the vehicle. Copter uses a simple “tin can border” as described in [Simple GeoFence](#). When the radius or maximum altitude is breached, Copter returns to the launch point and/or lands.

The fence is enabled (and its type selected) using the [Copter Fence Parameters](#). You can list the fence parameters with `param show`:

```
GUIDED> param show fence*
GUIDED> FENCE_ACTION      1.000000
FENCE_ALT_MAX    100.000000
FENCE_ENABLE     0.000000
FENCE_MARGIN     2.000000
FENCE_RADIUS     150.000000
```

FENCE_TYPE	3.000000
------------	----------

The fence has an altitude boundary of 100 metres (`FENCE_ALT_MAX`) and is bound by a circle of radius `FENCE_RADIUS` around the home location. The `FENCE_TYPE=3` means that both the radius and altitude are used (you can change the type to other numbers have an altitude-only or circle only fence - or none at all).

The fence is initially disabled (`FENCE_ENABLE=0`). To turn it on we set the value to one:

```
GUIDED> param set fence_enable 1
```

When we fly outside the radius the mode changes to RTL (return to land). If for some reason we travel further out by the value of `FENCE_MARGIN`, then the vehicle will simply land.

Testing the vehicle¶

MAVProxy allows you to list all the parameters affecting the vehicle and simulation using `param show *`, and to set any parameter using: `param set PARAM_NAME VALUE`. In addition to affecting the vehicle itself some parameters simulate the performance/failure of specific hardware components and the environment (for example, the wind). These can be listed using: `:ref:`param show sim``. The topic [Using SITL for ArduPilot Testing <using-sitl-for-ardupilot-testing>](#) explains more about how you can test using SITL.

## Using SITL for ArduPilot Testing

This article describes how to perform a number of common ArduPilot testing tasks in [SITL](#) using [MAVProxy](#).

Overview¶

The [SITL \(Software In The Loop\)](#) simulator is a build of the ArduPilot code which allows you to run [Plane](#), [Copter](#) or [Rover](#) without any hardware. It can be built and run on [Windows](#) or [Linux](#), and can be run on Mac OSX (or the other platforms) in a virtual machine with Linux installed.

Using SITL is just like using a real vehicle: you can connect to the (simulated) vehicle using the Ground Control Station (GCS) of your choice (or even multiple ground stations), take off, change flight modes, make guided or automatic missions, and land. The main difference is that in addition to being able to configure the vehicle, *simulator-specific* parameters allow you to configure the *physical environment* (for example wind speed and direction) and also to simulate failure of different components. This means that SITL is the perfect environment to test bug fixes and other changes to the autopilot, failure modes, and DroneKit-Python apps.

This article explains some of the more important parameters that you can set to change the environment, simulate failure modes, and configure the vehicle with optional components. It also explains how to [connect to different GSCs](#).

Tip

If you're just getting started with MAVProxy and SITL you may wish to start by reading the [Copter SITL/MAVProxy Tutorial](#) (or equivalent tutorials for the other vehicles).

Note

These instructions generally use [MAVProxy](#) to describe operations (e.g. setting parameters) because it presents a simple and consistent command-line interface (removing the need to describe a GSC-specific UI layout). There is no reason the same operations cannot be performed in *Mission Planner* (through the *Full Parameters List*) or any other GSC.

Setting vehicle start location¶

You can start the simulator with the vehicle at a particular location by calling `sim_vehicle.py` with the `-L` parameter and a named location in the `ardupilot/Tools/autotest/locations.txt` file.

For example, to start Copter in *Ballarat* (a named location in **locations.txt**) call:

```
cd ArduCopter
sim_vehicle.py -j4 -L Ballarat --console --map
```

#### Note

You can add your own locations to the file. The order is Lat,Lng,Alt,Heading where alt is MSL and in meters, and heading is degrees. If you need to use a location regularly then consider adding it to the project via a pull request.

#### Loading a parameter set¶

When starting SITL the first time, the device may be configured with “unforgiving” parameters. Typically you will want to replace these with values that simulate more realistic vehicle and environment conditions. Useful parameter sets are provided in the autotest source for [Copter](#), [Plane](#), and [Rover](#).

#### Tip

This only needs to be done once. After loading the parameters are stored in the simulated EEPROM. The MAVProxy commands to load the parameters for Copter, Rover and Plane (assuming the present working directory is a vehicle directory like `/ardupilot/ArduCopter/`) are shown below:

```
param load ..\Tools\autotest\default_params\copter.parm
```

```
param load ..\Tools\autotest\default_params\plane.parm
```

```
param load ..\Tools\autotest\default_params\rover.parm
```

You can re-load the parameters later if you choose, or revert to the default parameters by starting SITL (`sim_vehicle.py`) with the `-w` flag.

Parameters can also be saved. For example, to save the parameters into the present working directory you might do:

```
param save ./myparms.parm
```

#### Setting parameters¶

Many of the following tasks involve setting parameter values over MAVLink, which you do using the `param set` command as shown:

```
param set PARAMETERNAME VALUE
```

All available parameters can be listed using `param show`. The SITL-specific parameters start with `SIM_`, and can be obtained using:

```
param show SIM_*
```

#### Tip

When you change a parameter the value remains in the virtual EEPROM after you restart SITL.

Remember to change it back if you don't want it any more (or [reload/reset the parameters](#)).

#### Testing RC failsafe¶

To test the behaviour of ArduPilot when you lose remote control (RC), set the parameter `SIM_RC_FAIL=1`, as shown:

```
param set SIM_RC_FAIL 1
```

This simulates the complete loss of RC input. If you just want to simulate low throttle (below throttle failsafe level) then you can do that with the RC command:

```
rc 3 900
```

#### Testing GPS failure¶

To test losing GPS lock, use `SIM_GPS_DISABLE`:

```
param set SIM_GPS_DISABLE 1
```

You can also enable/disable a 2nd GPS using `SIM_GPS2_ENABLE`.

#### Testing the effects of vibration¶

To test the vehicle's reaction to vibration, use `SIM_ACC_RND`. The example below adds 3 m/s/s acceleration noise:

```
param set SIM_ACC_RND 3
```

#### Testing the effects of wind¶

The wind direction, speed and turbulence can be changed to test their effect on flight behaviour. The following settings changes the wind so that it blows towards the South at a speed of 10 m/s.

```
param set SIM_WIND_DIR 180
param set SIM_WIND_SPD 10
```

To see other wind parameters do:

```
param show sim_wind*
```

#### Note

At time of writing the wind parameters only appear to work for Plane.

#### Adding a virtual gimbal¶

SITL can simulate a virtual gimbal.

#### Note

Gimbal simulation causes SITL to start sending `MOUNT_STATUS` messages. These messages contain the orientation according to the last commands sent to the gimbal, not actual measured values. As a result, it is possible that the true gimbal position will not match - i.e. a command might be ignored or the gimbal might be moved manually. Changes are not visible in Mission Planner.

First start the simulator and use the following commands to set up the gimbal mount:

```
# Specify a servo-based mount:
param set MNT_TYPE 1

# Set RC output 6 as pan servo:
```

```
param set RC6_FUNCTION 6
# Set RC output 8 as roll servo:
param set RC7_FUNCTION 8
```

Then stop and re-launch SITL with the `-M` flag:

```
sim_vehicle.py -M
```

Adding a virtual rangefinder¶

SITL can simulate an analog rangefinder, which is very useful for developing flight modes that can use a rangefinder. To set it up use the following commands:

```
param set SIM SONAR_SCALE 10
param set RNGFND_TYPE 1
param set RNGFND_SCALING 10
param set RNGFND_PIN 0
param set RNGFND_MAX_CM 5000

# Enable rangefinder for Landing (Plane only!)
param set RNGFND_LANDING 1
```

The above commands will setup an analog rangefinder with a maximum range of 50 meters (the 50m comes from an analog voltage range of 0 to 5V, and a scaling of 10). After making the above changes you need to restart SITL.

Then to test it try this:

```
module load graph
graph RANGEFINDER.distance
```

Then try a flight and see if the graph shows you the rangefinder distance.

Tip

You can also use the following commands to graph rangefinder information (defined as *MAVProxy* aliases):

- `grangealt` - graph rangefinder distance and relative altitude.
- `grangev` - rangefinder voltage
- `grange` - graph “rangefinder\_roll”

Adding a virtual optical flow sensor¶

You can add a virtual optical flow sensor like this:

```
param set SIM_FLOW_ENABLE 1
param set FLOW_ENABLE 1
```

Then restart SITL. After setting it up try this:

```
module load graph
graph OPTICAL_FLOW.flow_x OPTICAL_FLOW.flow_y
```

Go for a flight and see if you get reasonable data.

Accessing log files¶

SITL supports both tlogs and DF logs (same as other types of ArduPilot ports). The DF logs are stored in

a “logs” subdirectory in the directory where you start SITL. You can also access the DF logs via MAVLink using a GCS, but directly accessing them in the logs/ directory is usually more convenient.

To keep your tlogs organised it is recommended you start SITL using the “–aircraft NAME” option. That will create a subdirectory called NAME which will have flight logs organised by date. Each flight will get its own directory, and will include the parameters for the flight plus any downloaded waypoints and rally points.

#### Graphing vehicle state¶

MAVProxy allows you to create graphs of vehicle state. Numerous aliases have been created for useful graph types in the *MAVProxy* initialisation file (**mavinit.scr**). These all start with “g” and include `gtrackerror`, `gaccel` etc.

#### Using a joystick¶

It can be useful to use a joystick for input in SITL. The joystick can be a real RC transmitter with a USB dongle for the trainer port, or something like the RealFlight interlink controller or a wide range of other joystick devices.

Before you use the joystick support you need to remove a debug statements from the python-pygame joystick driver on Linux. If you don’t then you will see lots of debug output like this:

```
SDL_JoystickGetAxis value:-32768:
```

To remove this debug line run this command:

```
sudo sed -i 's/SDL_JoystickGetAxis value/\x00DL_JoystickGetAxis value/g' /usr/lib/python2.7/dist-packages/pygame/joystick.so
```

note that this needs to be one long command line. Ignore the line wrapping in the wiki.

Then to use the joystick run:

```
module load joystick
```

If you want to add support for a new joystick type then you need to edit the [mavproxy\\_joystick module](#)

#### Using real serial devices¶

Sometimes it is useful to use a real serial device in SITL. This makes it possible to connect SITL to a real GPS for GPS device driver development, or connect it to a real OSD device for testing an OSD.

To use a real serial device you can use a command like this:

```
sim_vehicle.py -A "--uartB=uart:/dev/ttyUSB0" --console --map
```

what that does it pass the –uartB argument to the ardupilot code, telling it to use /dev/ttyUSB0 instead of the normal internal simulated GPS for the 2nd UART.

Any of the 5 UARTs can be configured in this way, using uartA to uartE.

Similar to this if you were running a vehicle in SITL via cygwin on Microsoft Windows and you wanted to send the mavlink output through a connected radio on COM16 to AntennaTracker you can use a command like this - note under cygwin comm ports are ttyS and they start at 0 so 15 is equivalent to COM16:

```
sim_vehicle.py -A "--uartC=uart:/dev/ttyS15" --console --map
```

### Changing the speed of the simulation¶

Most of the simulator backends support changing the speed while running. Just set the SIM\_SPEEDUP parameter as needed. A value of 1 means normal wall-clock time. A value of 5 means 5x realtime. A value of 0.1 means 1/10th of real time.

### Testing Compass Calibration¶

A quick way to test compass calibration in SITL is with the “calibration” vehicle model. To use this with plane do this:

```
sim_vehicle.py -j4 -D -f plane --model calibration --console --map
```

then do:

```
servo set 5 1250
```

This will start the vehicle moving through a “compass dance”. You can start a compass calibration to test changes to the calibrator code. Using this in combination with the SIM\_SPEEDUP parameter can be useful.

The calibration vehicle module has a lot of other features too. See

<http://guludo.me/posts/2016/05/27/compass-calibration-progress-with-geodesic-sections-in-ardupilot/> for details.

### Connecting other/additional ground stations¶

SITL can connect to multiple ground stations by using *MAVProxy* to forward UDP packets to the GCSs network address. Alternatively SITL can connect to a GCS over TCP/IP without using *MAVProxy*.

### SITL with MAVProxy (UDP)¶

SITL can connect to multiple ground stations by using *MAVProxy* to forward UDP packets to the GCSs network address (for example, forwarding to another Windows box or Android tablet on your local network). The simulated vehicle can then be controlled and viewed through any attached GCS.

First find the IP address of the machine running the GCS. How you get the address is platform dependent (on Windows you can use the ‘ipconfig’ command to find the computer’s address).

Assuming the IP address of the GCS is 192.168.14.82, you would add this address/port as a *MAVProxy* output using:

```
output add 192.168.14.82:14550
```

The GCS would then connect to SITL by listening on that UDP port. The method for connecting will be GCS specific (we show [how to connect for Mission Planner](#) below).

#### Tip

If you’re running the GCS on the **same machine** as SITL then an appropriate output may already exist. Check this by calling `output` on the *MAVProxy* command prompt.

```
GUIDED> output
GUIDED> 2 outputs
0: 127.0.0.1:14550
1: 127.0.0.1:14551
```

In this case we can connect a GCS running on the same machine to UDP port 14550 or 14551. We can choose to connect another GCS to the remaining port, and add more ports if needed.

**SITL without MAVProxy (TCP)¶**

It is also possible to interact with SITL over TCP/IP by starting it using `vehicle_name.elf` (e.g. `/ArduCopter/ArduCopter.elf`). MAVProxy is not needed when using this method.

Run the file in the *Cygwin Terminal*, specifying a home position and vehicle model as shown below:

```
$ ./ArduCopter.elf --home -35,149,584,270 --model quad
Started model quad at -35,149,584,270 at speed 1.0
Starting sketch 'ArduCopter'
Starting SITL input
bind port 5760 for 0
Serial port 0 on TCP port 5760
Waiting for connection ....
```

The command output shows that you can connect to SITL using TCP/IP at the network address of the machine **SITL is running on** at port 5760.

Tip

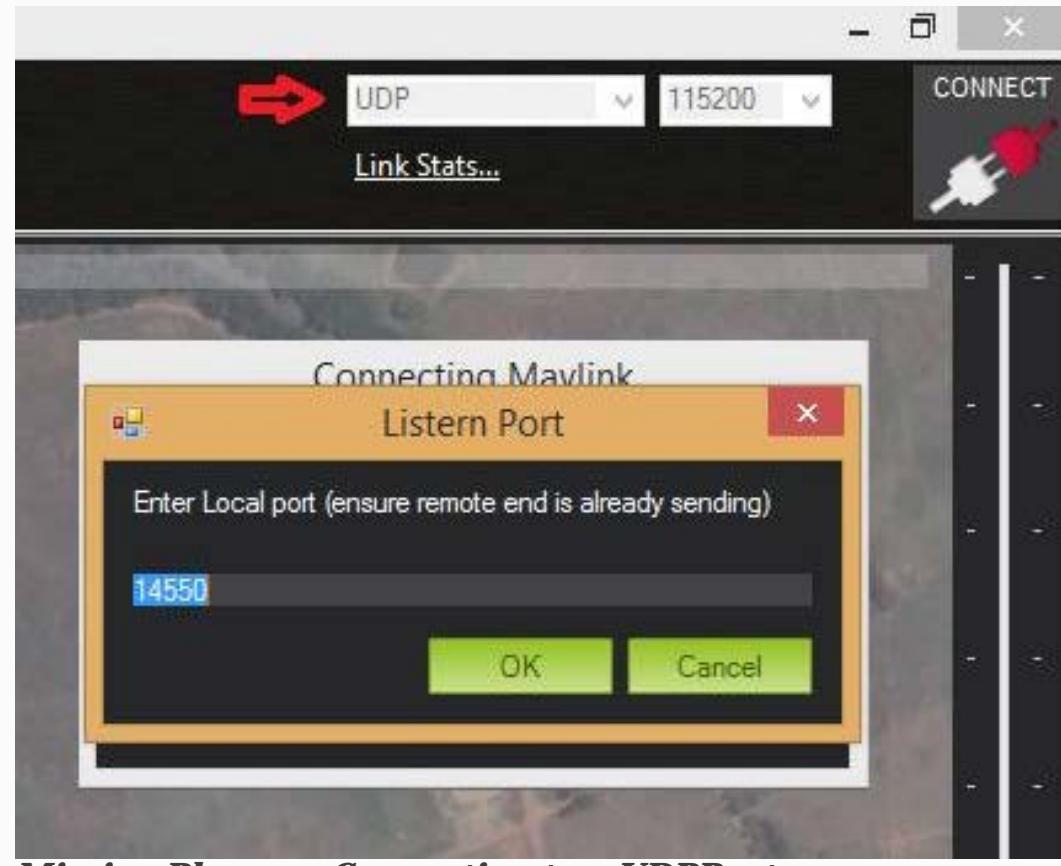
**ArduCopter.elf** has other startup options, which you can use using the `-h` command line parameter:

```
./ArduCopter.elf -h
```

**Connecting Mission Planner (UDP)¶**

First set up SITL to [output UDP packets to the address/port of the computer running \\*Mission Planner\\*](#).

In *Mission Planner* listen to the specific UDP port by selecting **UDP** and then the **Connect** button. Enter the port to listen on (the default port number of 14550 should be correct if SITL is running on the same computer).



***Mission Planner: Connecting to a UDPPort***

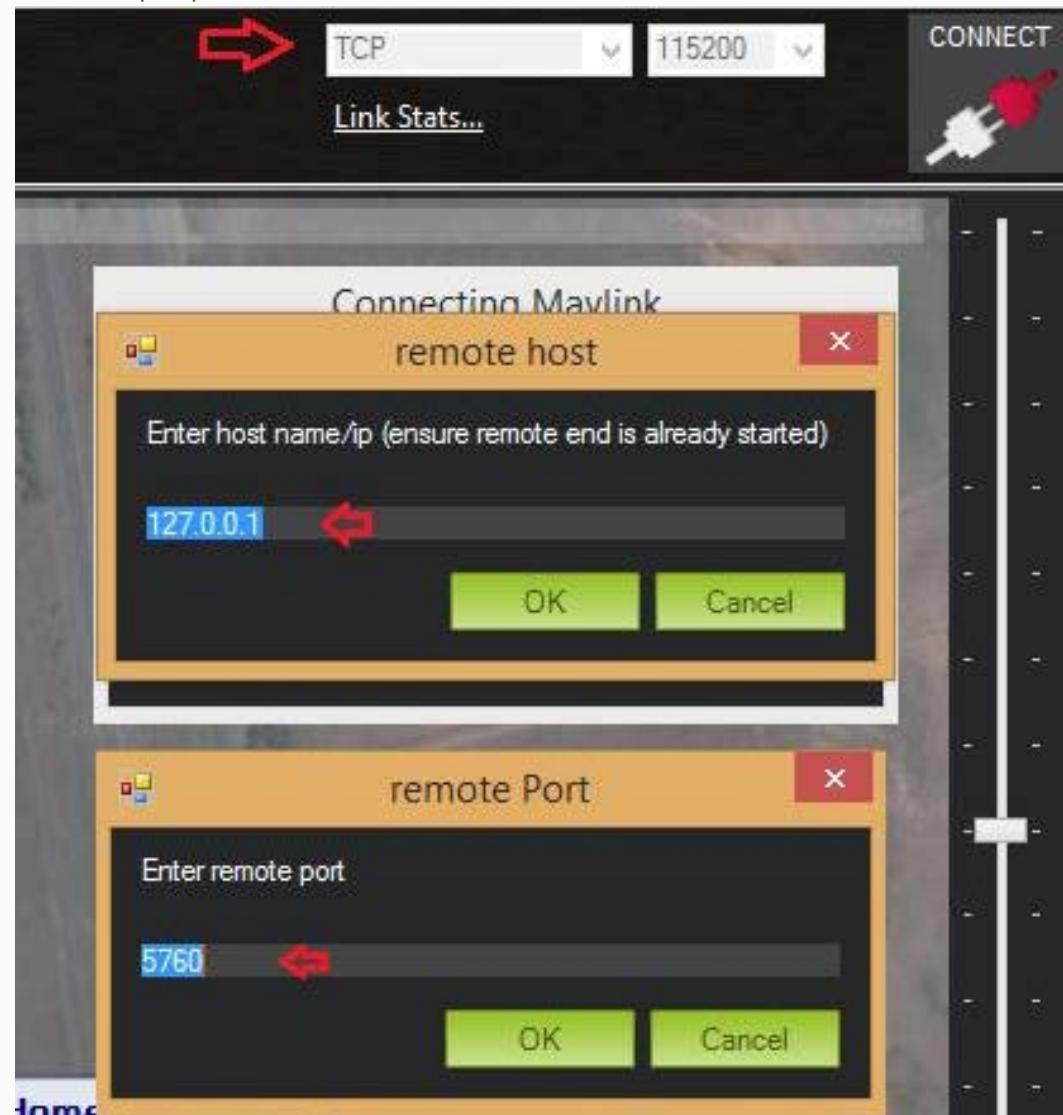
Connecting to Mission Planner (TCP)¶

First set up SITL [for use with TCP](#).

In *Mission Planner* connect to SITL by selecting **TCP** and then the **Connect** button. Enter the *remote host* and *remote Port* of the machine running SITL. *Mission Planner* will then connect and can be used just as before.

#### Tip

If SITL is running on the same machine as *Mission Planner* you can click through the *remote host* and *remote Port* prompts as these default to the correct values.



## ***Mission Planner: Connecting to SITL using TCP***

### **Using Gazebo Simulator with SITL**

This article explains how to use Gazebo <http://gazebosim.org/> as an external simulator for Copter.

#### Overview¶

Gazebo is a well-known and respected robotics simulator. We will be compiling Gazebo from source, because no current release has built-in support for ArduCopter.

#### Warning

Gazebo support is still under development, as of October 2016. If it seems that these instructions are outdated, please open an issue on the [ardupilot\\_wiki](#) github.

### Tip

Gazebo is particularly useful for defining autonomous indoor flights or swarms.

#### Preconditions

We recommend Ubuntu starting from 14.04.2 (it is working on 16.04 but untest on 16.10) as this is the platform used for testing this approach and is also known to be compatible with SITL.

#### Compiling and installing Gazebo

We will be using a standard version of ArduPilot but a custom fork of Gazebo, until the gazebo plugin gets merge into Gazebo-master.

These instructions are derived from [http://gazebosim.org/tutorials?tut=install\\_from\\_source](http://gazebosim.org/tutorials?tut=install_from_source)

Setup your computer to accept software from [packages.osrfoundation.org](http://packages.osrfoundation.org).

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
sudo apt-get update
sudo apt-get install build-essential cmake git libboost-all-dev mercurial libcegui-mk2-dev libopenal-dev
libswscale-dev libavformat-dev libavcodec-dev libltdl3-dev libqwt-dev ruby libusb-1.0-0-dev libbullet-dev
libhdf5-dev libgraphviz-dev libgdal-dev
```

### Install dependencies

```
wget https://bitbucket.org/osrf/release-tools/raw/default/jenkins-scripts/lib/dependencies_archive.sh -O /tmp/dependencies.sh
ROS_DISTRO=dummy . /tmp/dependencies.sh
sudo apt-get install $(sed 's:\\\ ::g' <<< $GAZEBO_BASE_DEPENDENCIES) $(sed 's:\\\ ::g' <<< $BASE_DEPENDENCIES)
```

### Remove old Gazebo versions

```
sudo apt-get remove .*gazebo.* .*sdformat.* .*ignition-math.*
```

### Make a gazebo workspace

```
mkdir ~/gazebo_ws
cd gazebo_ws/
```

### Build and install Ignition Maths

```
hg clone https://bitbucket.org/ignitionrobotics/ign-math ~/gazebo_ws/ign-math
cd ~/gazebo_ws/ign-math
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
make -j4
sudo make install
```

### Build and install Ignition Msgs

```
hg clone https://bitbucket.org/ignitionrobotics/ign-msgs ~/gazebo_ws/ign-msgs
cd ~/gazebo_ws/ign-msgs
mkdir build
cd build
```

```
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
make -j4
sudo make install
```

### Build and install Ignition Tools

```
hg clone https://bitbucket.org/ignitionrobotics/ign-tools ~/gazebo_ws/ign-tools
cd ~/gazebo_ws/ign-tools
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
make -j4
sudo make install
```

### Build and install SDFormat

```
hg clone https://bitbucket.org/osrf/sdformat ~/gazebo_ws/sdformat
cd ~/gazebo_ws/sdformat
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
make -j4
sudo make install
```

### Build and install Gazebo

```
hg clone https://bitbucket.org/osrf/gazebo ~/gazebo_ws/gazebo
cd ~/gazebo_ws/gazebo
hg checkout ardupilot
mkdir build
cd build
cmake ../
make -j4 # NOTE: This will take a long time!
sudo make install
```

Now try running Gazebo by typing `gazebo`. If it works, you're done. If it says

```
gazebo: error while loading shared libraries: libgazebo_common.so.1: cannot open shared object file: No such file or directory
```

Then find the file `libgazebo_common.so.1`, probably under `/usr/local/lib` or `/usr/local/lib/x86_64-linux-gnu`, and then add it like so:

```
echo '<insert directory here>' | sudo tee /etc/ld.so.conf.d/gazebo.conf
sudo ldconfig
```

### Note

Compiling Gazebo from source will not be necessary once this pull request gets merged:

<https://bitbucket.org/osrf/gazebo/pull-requests/2450/ardupilot-refactor-and-minor-improvements/diff>

Installing Custom Gazebo Models¶

We will also need to get a gazebo model of a quadcopter.

```
hg clone https://bitbucket.org/osrf/gazebo_models ~/gazebo_ws/gazebo_models
cd ~/gazebo_ws/gazebo_models
hg checkout zephyr_demos
echo 'export GAZEBO_MODEL_PATH=~/gazebo_models' >> ~/.bashrc
source ~/.bashrc
```

**Note**

This step will not be necessary once this pull request gets merged:

[https://bitbucket.org/osrf/gazebo\\_models/pull-requests/221/zephyr\\_demos/diff](https://bitbucket.org/osrf/gazebo_models/pull-requests/221/zephyr_demos/diff)

Set up PATH to build tools¶

If you have not already done so, you need to set up the PATH to the build tools (located in `/ardupilot/Tools/autotest`) so that the build system can find `sim_vehicle.py`.

Navigate the file system to the home directory and open the `.bashrc` file. Add the following line to the end of `.bashrc`:

```
export PATH=$PATH:$HOME/ardupilot/Tools/autotest
```

**Note**

Use your own path to ardupilot folder in the line above!

Start the Simulator¶

In one terminal, enter the ArduCopter directory and start the SITL simulation:

```
cd ~/ardupilot/ArduCopter
sim_vehicle.py -f gazebo-iris
```

In another terminal start Gazebo:

```
cd ~/gazebo
gazebo --verbose worlds/iris_standoff_demo.world
```

If all works well, you should see this:

**Note**

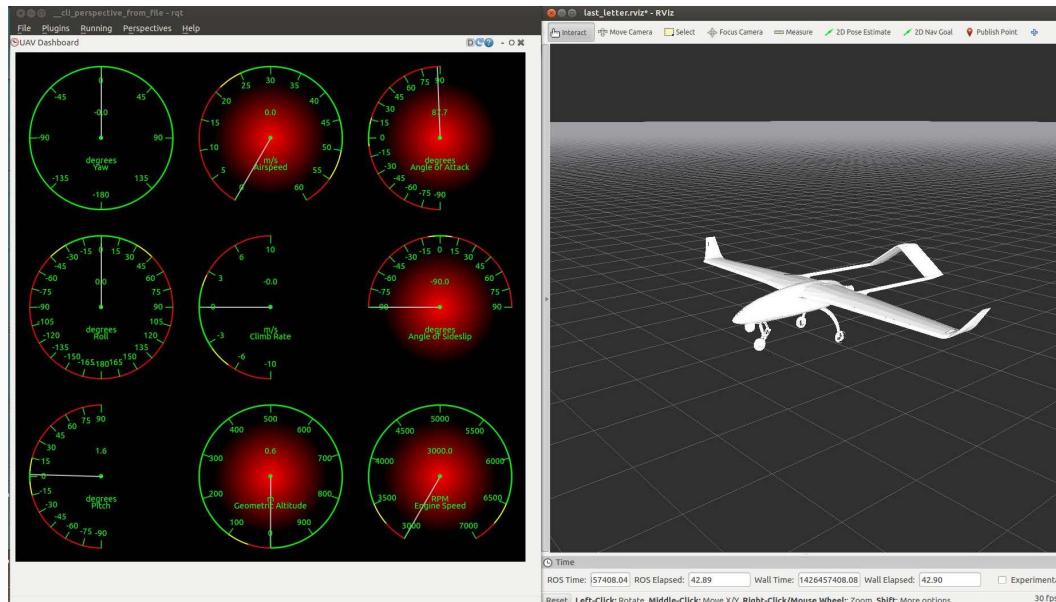
ROS is commonly used together with Gazebo, but this is out of the scope of this article. If you are using ROS, some packages to consider using are: - `mavros` (for sending and receiving mavlink packets) - `ros_gazebo_camera` (for publishing Gazebo's virtual camera stream to a ROS topic)

### Using `last_letter` as an External SITL Simulator

This article explains how to run `SITL` for Plane using `last_letter` as the flight simulation engine (instead of `JSBSim`).

**Note**

Currently the `last_letter` simulator is in early development stage and also only supports fixed-wing aircraft. Expect frequent updates and changes in its architecture. Up-to-date documentation on `last_letter` is always found at its Github page.



## Overview

Currently, the default simulation software for Plane SITL is [JSBSim](#). While this is a renowned simulation solution for fixed-wing aircraft, it is primarily targeted towards full-scale vehicles and focuses on relevant features such as fuel management, variable pitch drives and oleos. After many years of life it is difficult to modify to cater for UAV-specific applications. Moreover it is paired with [FlightGear](#) as the visualizer, which is a quite a demanding piece of software in terms of system resources.

In contrast, during the development of a robotic platform it is often more important to have easy access to the system states, be able to change simulator components easily, add new subsystems, and interface with other packages.

The [Robotics Operating System](#) (ROS) framework is a potential answer to this need. It is a software framework which resides on top of the operating system, orchestrating the execution of multiple programs and the communication between them. It provides the tools to compile programs written in C++ and Python and monitor their execution. It is specifically targeted for development of modular robotics applications by providing standardized message headers, universally useful system tools, logging capabilities, execution across multiple machines and most importantly a large codebase of user-submitted, stand-alone packages, fit for diverse applications.

*last\_letter* is a flight simulator written mostly in C++ under the ROS framework. It incorporates a modular description of the airplane model, modeling aerodynamics, propeller and motor dynamics, ground reactions, wind and gust disturbances and much more. Its structure allows for relatively easy coding of additional functions, and offers the potential for a unified simulator for all platforms, compatibility with other ROS packages and a modular, detailed simulation solution.

## Preconditions

### ROS version

*Last\_letter* is being developed under *ROS Indigo*, and this is the only supported version.

## Tip

*Last\_letter* was originally developed under *ROS Hydro*, and no code changes were required during the transition. While *ROS Hydro* is no longer supported, developers that need to use *last\_letter* on this version may find it still works.

### Operating System requirements

The selected operating system must be [compatible with ROS Indigo](#) in order to run *last\_letter*.

We recommend Ubuntu 14.04.2 as this is the platform used for developing *last\_letter* and is also known to be compatible with SITL. Ubuntu Linux 13.10 and 14.04 are also supported by both *ROS Indigo* and

SITL.

#### Installation¶

First install the Plane SITL on your Linux machine, as described [Setting up SITL on Linux](#).

Like any other set of ROS packages, *last\_letter* and its companion library packages **last\_letter\_msgs**, **uav\_utils**, **math\_utils** and **rqt\_dashboard** require ROS to be present on your system. If you have previous experience with ROS and have it on your system, installation of *last\_letter* is as simple as cloning the package files in your workspace. If not, the following instructions will guide you through the process.

#### ROS installation¶

Follow the [instructions on the ROS website](#) to install *ROS Indigo* in your machine.

#### Workspace creation¶

After installing ROS you need to create a ROS Workspace. This is the directory where custom software packages (packages which are not part of the official ROS repository) are created, copied or edited. This is also where the ROS compiler (also known as *catkin*) looks for your code.

Follow these [instructions to create your workspace](#). They instruct you to create your workspace under your home folder, but it can also be created elsewhere. Just be sure you have full rights to this folder, so a subfolder in your home folder is a good choice. Make sure `$ROS_PACKAGE_PATH` sees your **catkin\_ws/src** folder.

#### Tip

You probably don't want to source your workspace path every time you open a new console. To avoid this, add the following line to your `/home/<username>/.bashrc` file (assuming that you created your catkin workspace directly under your home folder).

```
source ~/catkin_ws/devel/setup.bash
```

#### Cloning the last\_letter simulator files¶

Execute the following commands in your console to download and compile the simulator (without the comments):

```
roscd
cd .. /src #Navigate in your ROS user source files directory
git clone https://github.com/Georacer/last_letter.git #Clone the simulator files
roscd
cd .. #Navigate in your ROS workspace
catkin_make #Compile the files
```

Compilation may take a while.

#### Testing everything is installed properly¶

In a console, run:

```
roslaunch last_letter launcher.launch
```

This should start the simulator and open a 3D simulation environment using an application called *RViz*. The *rqt\_gui* application will start as well; this is where the avionics instruments are displayed.

If everything went smoothly, you can close everything and proceed to interface *last\_letter* with Plane SITL.

#### Using last\_letter as the physics simulator for SITL¶

Follow the [Plane SITL instructions](#) to download and compile the Plane code. Make sure your local git branch is checked out on `master`.

When everything is done, enter the Plane directory and start the SITL simulation:

```
cd ~/ardupilot/ArduPlane
sim_vehicle.py -f last_letter --console --map
```

This will run SITL, Plane, MAVProxy and *last\_letter* along with RViz all in one go (the RViz visualizer may take several seconds to start). This procedure will automatically invoke the *last\_letter* launch file and set the `Plane` parameter to `true` so that the communication nodes between *last\_letter* and Plane are raised. If *last\_letter* communicates with Plane correctly, the MAVProxy console should display 3D satellite fix and track 10 satellites.

When you start sim\_vehicle.py an xterm will be created which launches ArduPilot and *last\_letter*. If you have troubles running the simulator then finding this (minimised) xterm and looking at the debug output there can be very helpful.

You can issue commands or RC overrides using MAVProxy as normal.

#### Missing features¶

At time of writing (17 March, 2015), *last\_letter* does not support the common initialization arguments (such as starting location). Moreover, the MAVProxy commands which would control the JSBSim simulator (such as the wind and pause commands) are not supported.

Currently, the only way to select the aircraft and alter its initial states is by editing the parameter files of the simulator, as described in [the corresponding manual page](#).

#### Running *last\_letter* in a virtual machine¶

*Last\_letter* has been successfully run within virtual machines using the configurations described in its [compatibility information](#). To test the performance of the simulator in a particular environment, look at the xterm window assigned for the simulator output: There should be a framerate message, updating every 5 seconds.

Currently, Plane and SITL are set to run at a nominal 500Hz. A frequency of over 480Hz indicates that the simulator is running at an acceptable speed.

## Using the CRRCSim simulator

You can optionally use the CRRCSim flight simulator with ArduPilot SITL. The main advantages of using CRRCSim is that it offers a lot more fixed wing models, and also offers a helicopter simulator.

#### Installing CRRCSim¶

The CRRCSim simulator has been modified for use by ArduPilot SITL. To install it on Linux you need to do the following:

Add dependencies

```
sudo apt-get install plib1.8.4-dev libjpeg-dev libsd12-dev
sudo apt-get install libportaudio-dev libcgal-dev #optional for audio/thermal support
```

To download code and build

```
git clone git://github.com/tridge/crrcsim-ardupilot.git
cd crrcsim-ardupilot
./autogen.sh
./configure
```

```
make
sudo make install
```

That will install crrcsim in /usr/local/bin. You may find you are missing some packages needed for CRRCSim at the configure stage.

### Running SITL with CRRCSim

Once crrcsim is installed you can launch it by running:

```
crrcsim -i APM
```

that starts CRRCSim with the APM protocol interface. You can then press ESCAPE to bring up the menu and choose an aircraft to simulate. Many of the aircraft will work with SITL, but for ones without motors (the gliders) you will need to choose a launch location on a slope.

For fixed wing testing it is recommended you start with the “Sport” aircraft. That simulates a small nitro sport aircraft.

For helicopter testing choose the “Heli-APM” model.

After you have launched CRRCSim you need to start SITL. For fixed wing testing use the “-f CRRCSim” option to sim\_vehicle.py:

```
cd ArduPlane
sim_vehicle.py -f CRRCSim --console --map
```

### Simulating a helicopter

For helicopter testing with CRRCSim use “-f CRRCSim-heli”

```
cd ArduCopter
sim_vehicle.py -f CRRCSim-heli --console --map
```

The helicopter will have the RSC speed on channel 8.

## SITL on Windows in a VMWare VM (Manual Setup)

This article explains how to *manually* set up the [SITL ArduPilot Simulator](#) in a virtual machine environment on Windows, and connect it to a Ground Control Station

### Tip

See [Setting up SITL using Vagrant](#) for a more automated setup process.

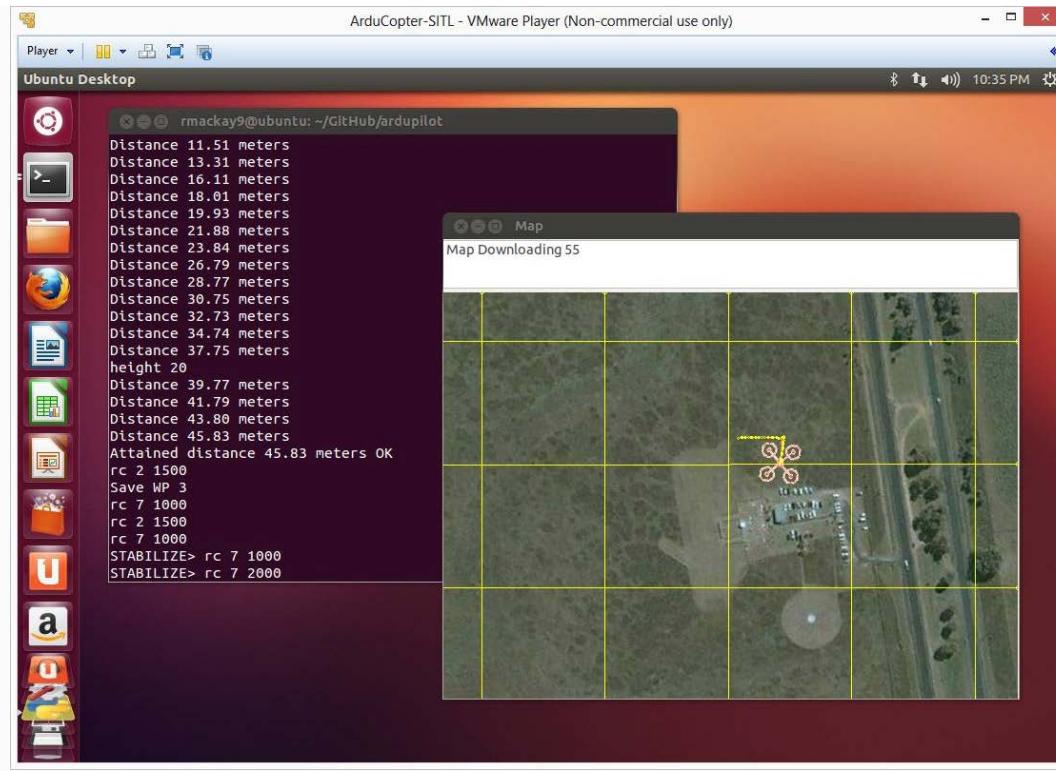
### Overview

The SITL (Software In The Loop) simulator allows you to run Plane, Copter or Rover without any hardware. The simulator runs the normal ArduPilot code as a native executable on a Linux PC. SITL can also be run within a virtual machine on Windows, Mac OSX or Linux.

This article shows how to manually set up SITL on a Linux environment hosted in a VMware virtual machine.

### Note

The instructions were tested on Windows 8 with VMware ver 7.1.0 build-2496824 and Ubuntu 14.10.



### Preconditions

Please note that before attempting this you should already have [downloaded the code](#) to your machine and be able to [build it with Arduino or Make](#).

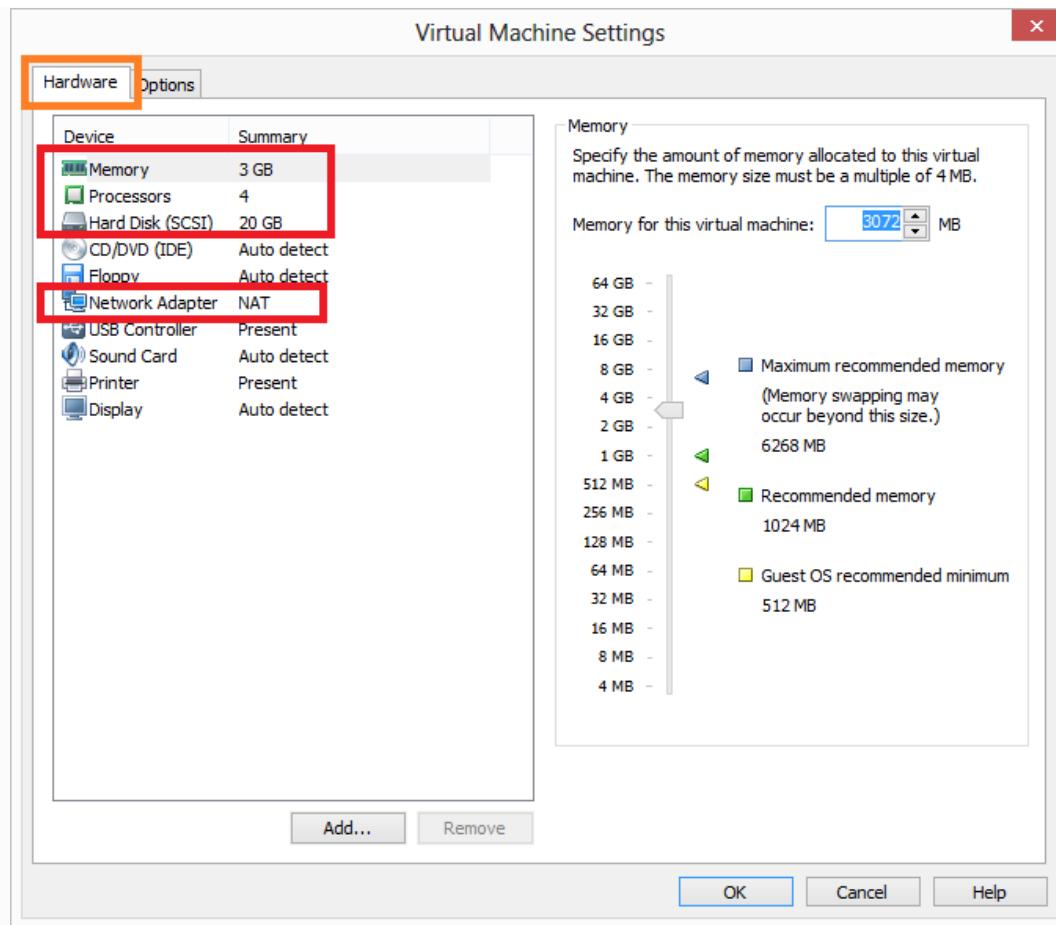
#### Step #1 - Install VMWare and create an Ubuntu Virtual Machine

1. Download and install [VMware](#) (look for VMware Player and VMware Player Plus for Windows)

2. Download the [Ubuntu iso](#)

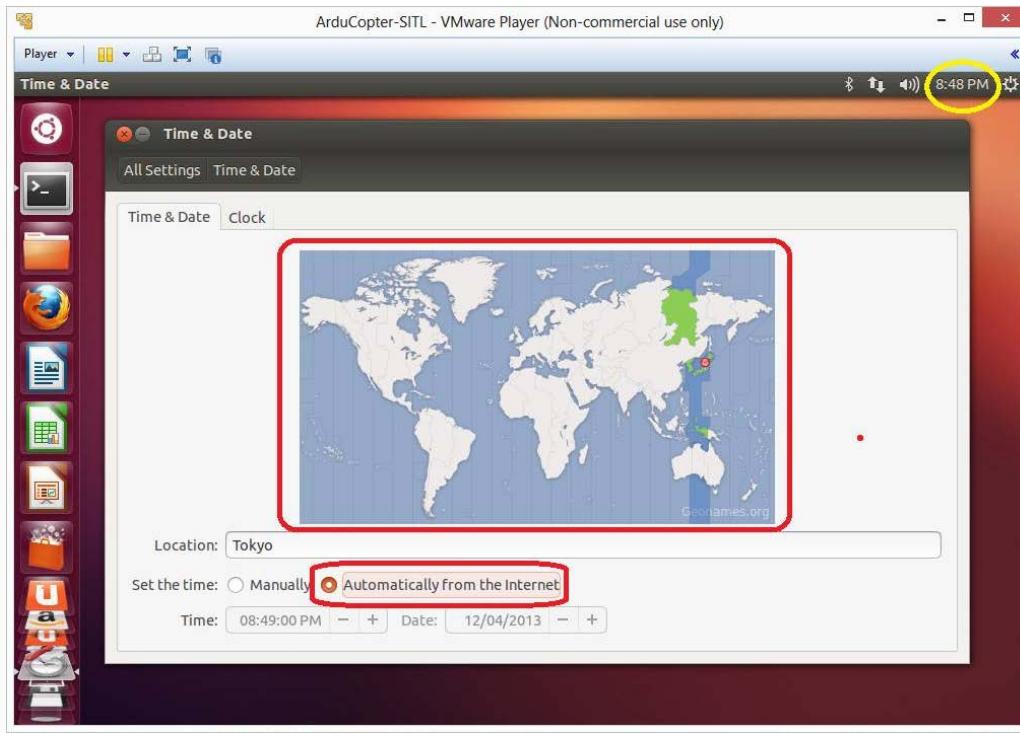
3. Start VMware and create new ubuntu machine by selecting Player > File > New Virtual Machine

- Enter your Full name, user name and password for the virtual machine. You will use these later when you log onto this virtual machine
- Name your virtual machine (i.e. Copter-SITL)
- Specify Disk Capacity - leave Max disk size as default 20GB, and "Split virtual disk into multiple files" checked
- On next page of "New Virtual Machine Wizard" click the "Customize Hardware.." button
- On the Hardware tab set Memory: 3GB, Processors: 4, Hard Disk: 20GB, Network Adapter: NAT

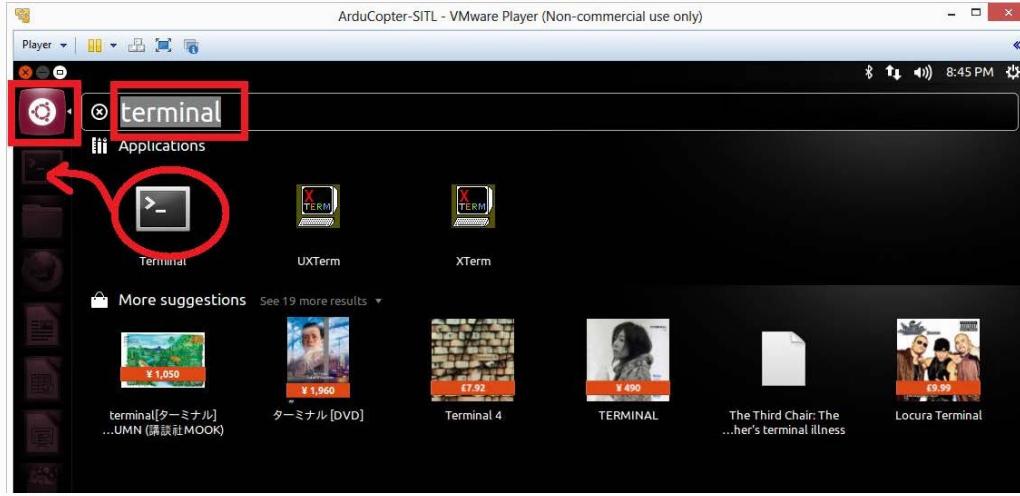


#### Step #2 - Start the VM for the first time ↗

1. Power on the machine by double clicking on item just created in the VMware Player
2. Press "No" to any questions like "Cannot connect to the XXX device because no corresponding device is available on the host"
3. Enter your password when the login screen appears
4. Say "no" to any options to upgrade versions
5. Open firefox and make sure it can access some web page like [www.ardupilot.org](http://www.ardupilot.org)
6. Set the clock by double clicking on the top right corner, select your location on the map and "Set the time:" to "Automatically from the internet"

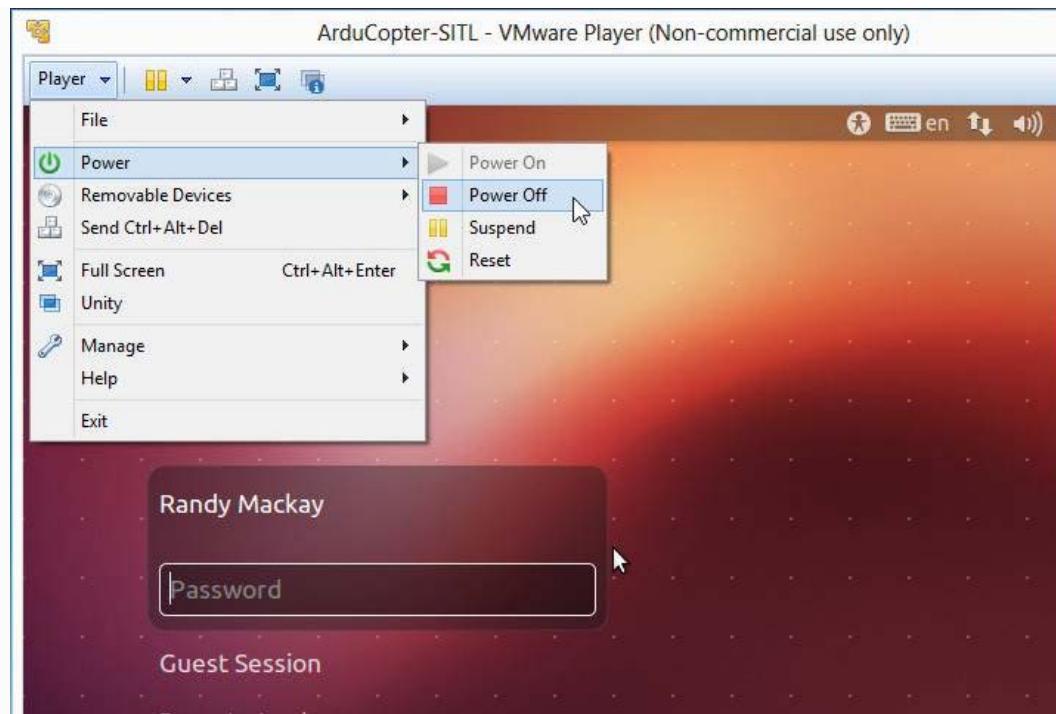


7. Setup a terminal short by clicking on the Dash Home icon on the top left, Type “terminal” and then drag the terminal applicatoin to the left start pane



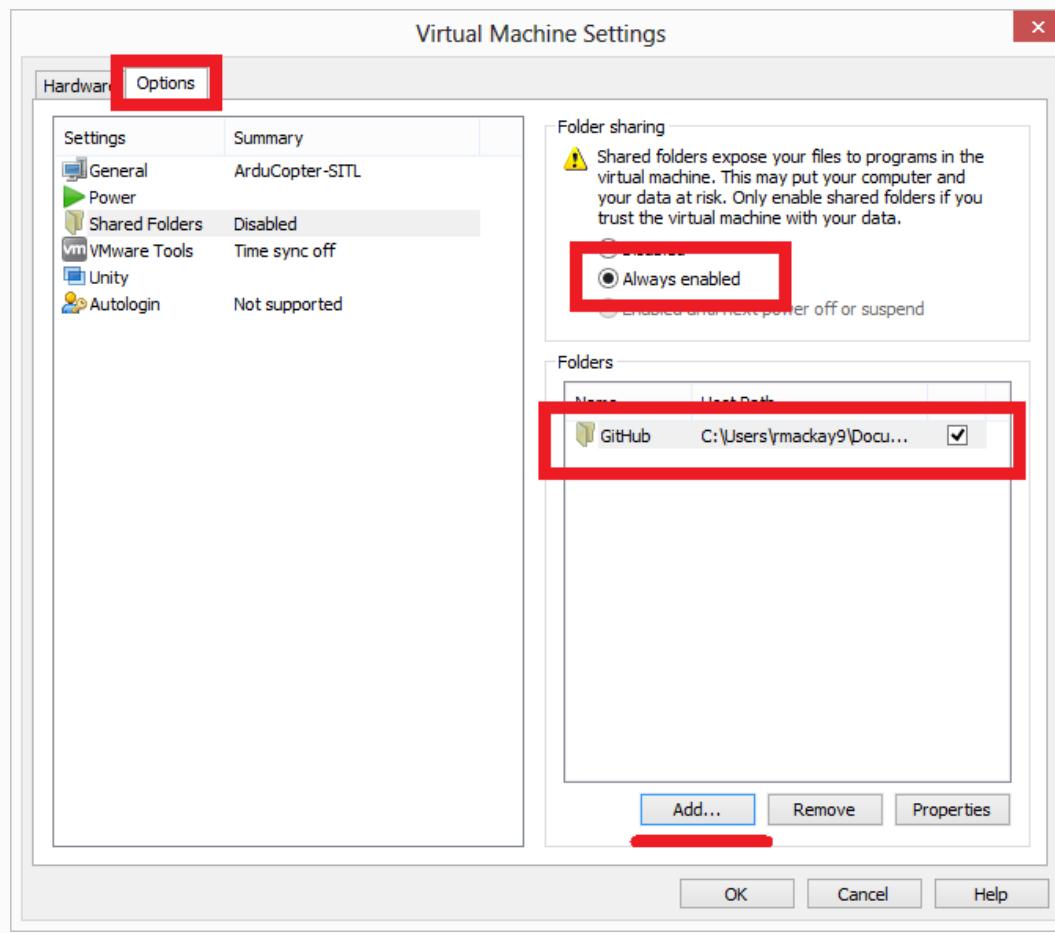
Step #3 - Setting up shared partition with Windows & Ubuntu Virtual Machine [¶](#)

1. Ensure the VM is powered down by select it's name, then select the drop-down beside the green play button and select “Power Off” if it’s not greyed



2. On VMware Player select Player > Manage > Virtual Machine Settings... > Options Tab > Shared Folders

- check "Always enabled", Add
- "Host Path" to folder one level above where you have installed ardupilot software
- check "Enable this share"



3. Power on the VM, enter your login information
4. Open Terminal window and type "ls /mnt/hgfs" and the share you set-up should be visible
5. type "ln -s /mnt/hgfs/<foldername>" (where <foldername> is replaced with the name of the folder you set-up) to create a symbolic link to the shared folder from your home directory

#### Step #4 - install packages on your VM¶

Note

Most of these same dependencies will be installed when you do the next step ([SITL instructions for Linux](#)).

Open up a terminal and type the following to update the list of packages in the software center:

```
sudo apt-get update
```

Then install the following packages (reply 'y' if it prompts you re additional disk space used)

```
sudo apt-get install python-dev dos2unix python-wxgtk2.8 python-matplotlib python-opencv python-pip g++  
g++-4.7 gawk git ccache  
  
sudo pip install pymavlink  
sudo pip install mavproxy
```

If you wish to run Plane you will also need to install these packages:

```
sudo apt-get install libexpat1-dev autoconf libtool automake
```

#### Step #5 - Follow the Linux instructions¶

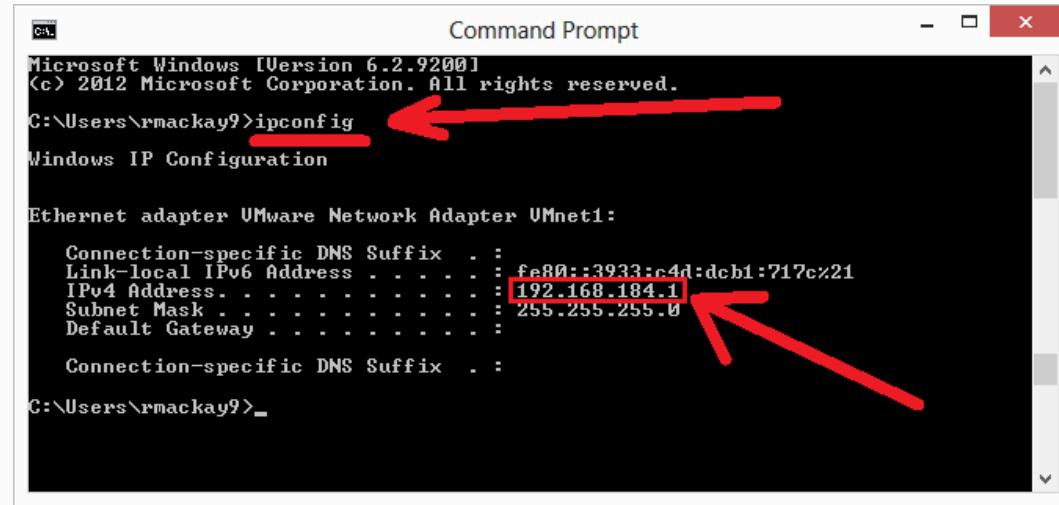
Now that you have a Linux VM you should follow the [SITL instructions for Linux](#)

#### Connecting with the Mission Planner¶

In addition to using the mavproxy ground station (the command line style ground station written in python) it should be possible to connect with the Mission Planner by appending the `--viewerip=XXX.X.X.X` to the end of the start up command where the Xs are replaced with your machine's IP address. This address can be found by typing "ipconfig" into a command prompt.

Note

You will likely see many more than one IP address listed so you may need to try a few different addresses to find one that works.



So for this example the following would then be entered into the terminal on the Ubuntu VM:

```
./Tools/autotest/autotest.py build.ArduCopter fly.ArduCopter --map --viewerip=192.168.184.1
```

Next connect with the mission planner after first setting the “COM Port” to “UDP”.



### Dataflash logs [¶](#)

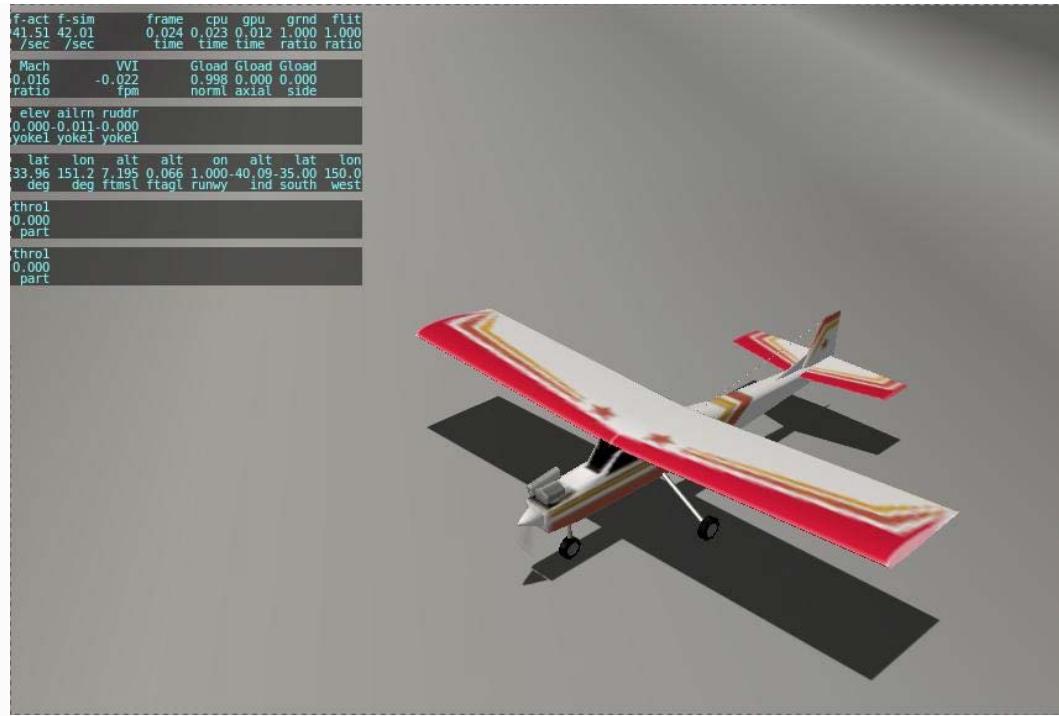
After the simulation is run, dataflash logs named “Copter.flashlog” or “CopterAVC.flashlog” will be created in the “buildlogs” directory. This directory is created at the same level as the ardupilot directory (i.e. the top level directly that holds the “Copter”, “Plane” and “libraries” directories). Because of the inconvenient name you’ll need to change the file extension to “.log” before opening in Mission Planner.

### Next steps [¶](#)

To get the most out of SITL we recommend you [Learn MavProxy](#).

The topic [Using SITL for ArduPilot Testing](#) explains how to use the simulator, and covers topics like how to use SITL with Ground Stations other than Mission Planner and MAVProxy.

## Using SITL with X-Plane 10



This article describes how to use X-Plane 10 as a simulation backend for ArduPilot [SITL](#).

### Overview

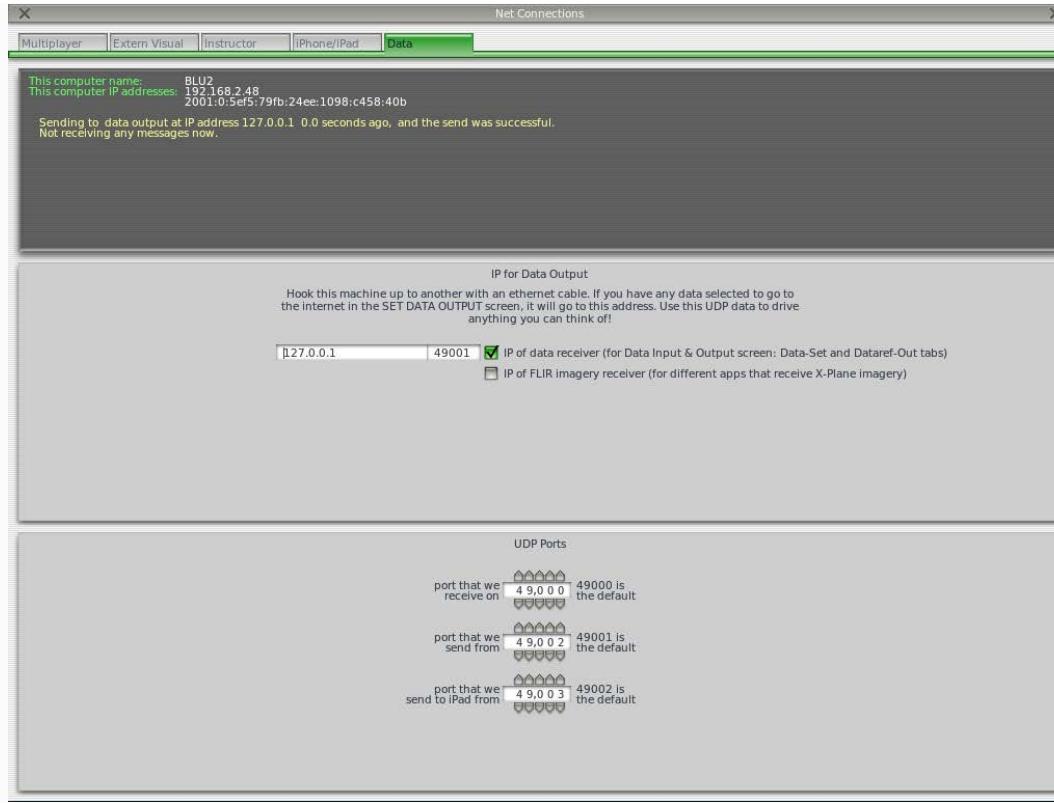
X-Plane 10 is a commercial flight simulator with a rich networking interface that allows it to be interfaced to other software. In this case we will be interfacing it to the ArduPilot SITL system, allowing ArduPilot to fly a wide variety of aircraft.

Using X-Plane with SITL is a good way to get some experience flying ArduPilot and learning how to use the ground control station. It can also be used to see how ArduPilot handles unusual aircraft and to develop support for aircraft features that may not be available in other simulator backends.

### Setup of X-Plane 10

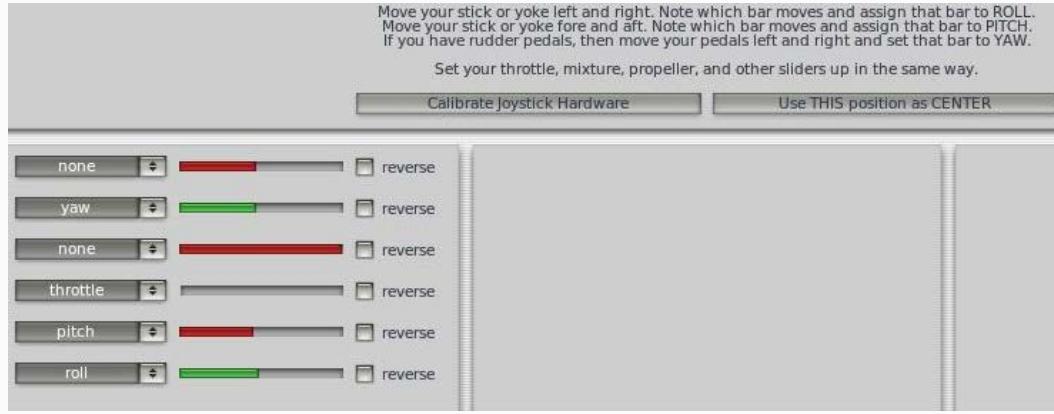
Before starting SITL the only thing you need to setup on X-Plane is the network data to send the sensor data to the IP address of the computer that will run ArduPilot. This can be the same computer that is running X-Plane (in which case you should use an IP address of 127.0.0.1) or it can be another computer on your network.

Go to the Settings -> Net Connections menu in X-Plane and then to the Data tab. Set the right IP address, and set the destination port number as 49001. Make sure that the receive port is 49000 (the default). If using loopback (ie. 127.0.0.1) then you also need to make sure the “port that we send from” is not 49001. In the example below 49002 is used.



If you have a joystick then you can configure the joystick for X-Plane. A joystick controlled by X-Plane will be available as R/C input when ArduPilot is in control of X-Plane, allowing you to fly the aircraft with the joystick in ArduPilot flight modes.

For joystick setup go to Settings -> Joystick and Equipment. You should setup controls for roll, pitch, yaw and throttle. Note that X-Plane has an unusual throttle setup where the bar is fully to the left at full throttle and fully to the right at zero throttle.



Right now you can't use the joystick for other than basic axes controls, so you can't use it for flight mode changes. We may be able to add support for that in the future.

### Starting SITL

There are three approaches to starting SITL with X-Plane depending on what you are wanting to do.

- running SITL from within MissionPlanner on Windows
- building SITL yourself and connecting from your favourite GCS
- building and running SITL using sim\_vehicle.py and MAVProxy

The first approach is good if you just want to test ArduPilot with SITL but you don't want to make changes to the code. MissionPlanner will download a build of ArduPilot SITL for Windows that is built each night

from git master.

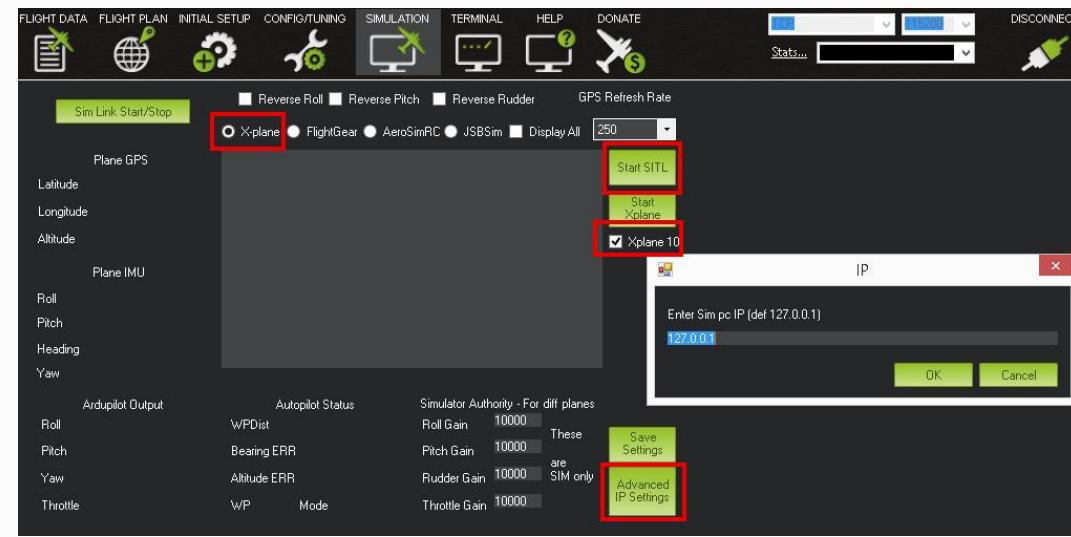
The second approach is good if you want to do ArduPilot development and try out code changes and you want to use a ground station of your choice. Any ground station that supports MAVLink over TCP can be used.

The third approach is good if you want the full capabilities of MAVProxy for ArduPilot SITL testing. MAVProxy has a rich graphing and control capability that is ideal for long term ArduPilot software development.

#### Using SITL from MissionPlanner

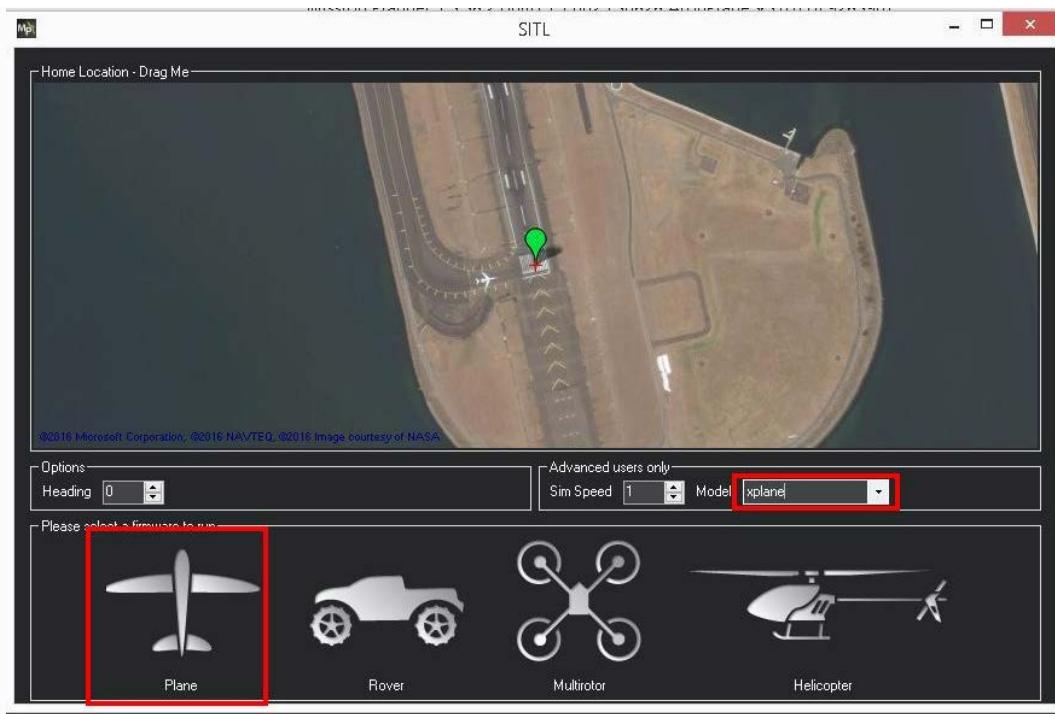
To start SITL directly from MissionPlanner you need to have a very recent version of MissionPlanner. Get the latest beta from the help screen.

Then go the SIMULATION tab:



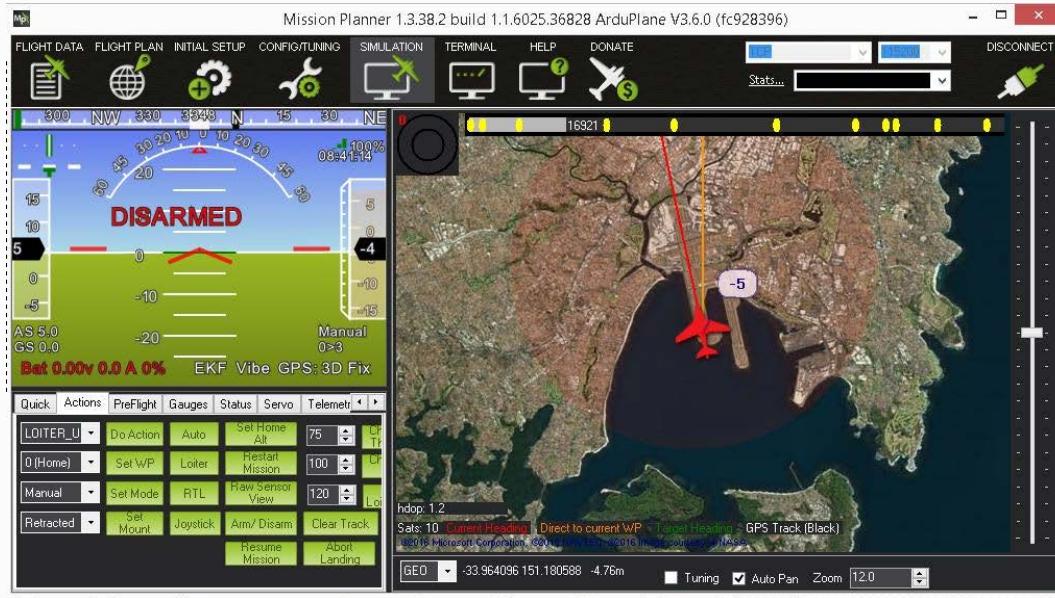
In the SIMULATION tab select X-plane and Xplane 10. Then select Advanced IP Settings an click through the IP addresses, set them to 127.0.0.1, with the default network ports.

Press "Start SITL"



In the SITL screen you need to select Model “xplane” and then select “Plane”. At the moment we only support fixed wing and helicopter aircraft in X-Plane with SITL. In the future we may support other aircraft types. See below for more information on flying a helicopter.

When you select “Plane” MissionPlanner will download a nightly build of ArduPilot SITL and will then launch SITL.



You then need to load an appropriate set of parameters for the aircraft (or setup the aircraft just like you would a real aircraft) and enjoy flying as usual with MissionPlanner.

When setting up the aircraft it is useful to use the joystick to move the control surfaces to make sure they are all going the right way. You can change channel direction in the normal way with ArduPilot parameters.

#### Using SITL with your own GCS

The second approach to running X-Plane 10 with SITL is to build ArduPilot SITL manually and then run it from the cygwin command line. You can then connect with your favourite GCS.

You should checkout the latest ArduPilot git tree in cygwin, and then change directory to the top "ardupilot" directory. Then run the following commands:

```
$ modules/waf/waf-light configure --board sitl
$ modules/waf/waf-light plane
$ build/sitl/bin/arduplane --model xplane
```

The screenshot shows a terminal window titled "tridge@blu2:~/ardupilot". The session starts with running the "configure" script for the SITL board, which performs various checks and sets up build paths. It then runs the "waf-light plane" command, which enters the build directory for SITL and performs a build. A summary of the build is provided, showing memory usage for different targets. Finally, the "arduplane" executable is run with the "--model xplane" option, starting the SITL simulation and opening a serial port for connection.

```
tridge@blu2:~/ardupilot(master)$ ./modules/waf/waf-light configure --board sitl
Setting top to : /home/tridge/ardupilot
Setting out to : /home/tridge/ardupilot/build
Setting board to : sitl
Checking for 'g++' (C++ compiler) : /usr/bin/g++
Checking for 'gcc' (C compiler) : /usr/bin/gcc
Checking for need to link with librt : not necessary
Checking for HAVE_CMATH_ISFINITE : yes
Checking for HAVE_CMATH_ISINFINITY : yes
Checking for HAVE_CMATH_ISNAN : yes
Checking for NEED_CMATH_ISFINITE_STD_NAMESPACE : yes
Checking for NEED_CMATH_ISINFINITY_STD_NAMESPACE : yes
Checking for NEED_CMATH_ISNAN_STD_NAMESPACE : yes
Checking for program 'python' : /usr/bin/python
Checking for python version : (2, 7, 10, 'final', 0)
Checking for program 'git' : /usr/bin/git
Checking for program 'size' : /usr/bin/size
Benchmarks : disabled
Unit tests : enabled
'configure' finished successfully (2.626s)
tridge@blu2:~/ardupilot(master)$ ./modules/waf/waf-light plane
Waf: Entering directory '/home/tridge/ardupilot/build/sitl'
Waf: Leaving directory '/home/tridge/ardupilot/build/sitl'

BUILD SUMMARY
Build directory: /home/tridge/ardupilot/build/sitl
Target      Text   Data   BSS Total
-----
bin/arduplane 1386257 41620 276 1428153
bin/arduplane-tri 1371701 41620 276 1413597
Build commands will be stored in build/sitl/compile_commands.json
'plane' finished successfully (4.704s)
tridge@blu2:~/ardupilot(master)$ ./build/sitl/bin/arduplane --model xplane
Waiting for XPlane data on UDP port 49001 and sending to port 49000
Started model xplane at -35.363261,149.165230,584,353 at speed 1.0
Starting sketch 'ArduPlane'
Starting SITL input
bind port 5760 for 0
Serial port 0 on TCP port 5760
Waiting for connection ....
```

That will start SITL and wait for a GCS to connect. You should connect on TCP port 5760 and configure ArduPilot as usual.

#### Using SITL with sim\_vehicle.py

The `sim_vehicle.py` script gives you a lot of options for launching all of the different simulation systems that work with ArduPilot, including X-Plane 10.

To use `sim_vehicle.py` you will need to install MAVProxy. If you are on Linux then make sure pip is installed and run:

```
$ pip install --upgrade pymavlink mavproxy
```

If you are on Windows then download and install MAVProxy from <http://firmware.ardupilot.org/Tools/MAVProxy/>

Then do a git checkout of ArduPilot master and change directory to the ArduPlane directory. I like to create a sub-directory for each aircraft I fly in SITL so that settings are remembered per-aircraft. If you want to do that then create a subdirectory in the ArduPlane directory and run `sim_vehicle.py` from there. In the following example I will be using the PT60 aircraft in X-Plane, so I create a PT60 directory:

```
$ cd ArduPlane
$ mkdir PT60
$ cd PT60
$ sim_vehicle.py -j4 -D -f xplane --console --map
```

### Flying a Helicopter [¶](#)

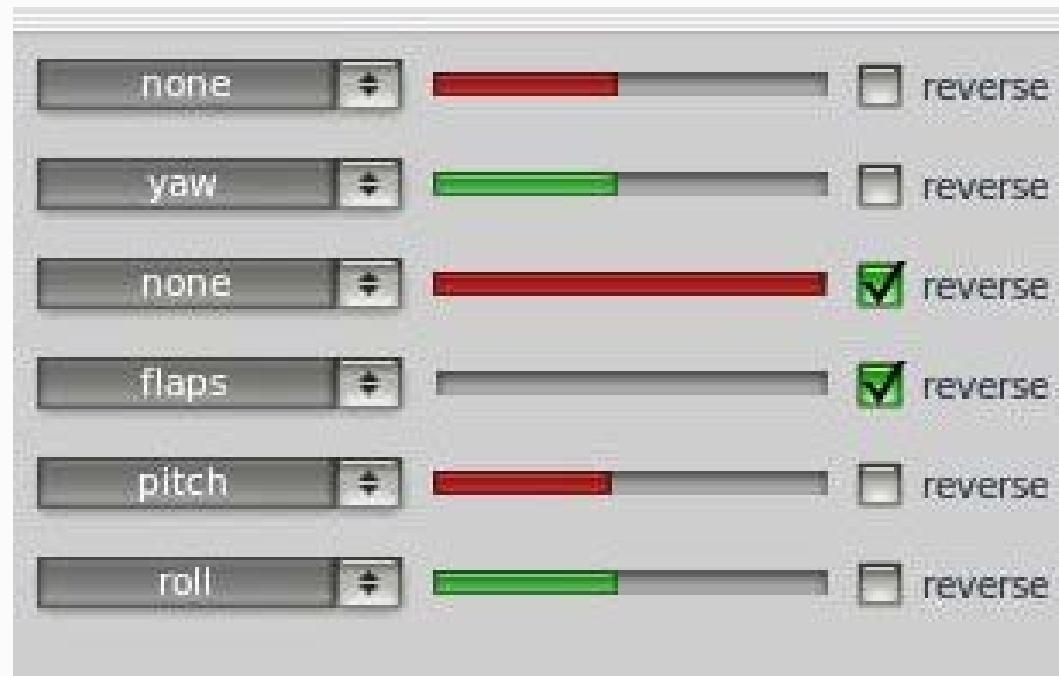
It is also possible to fly a helicopter with XPlane-10. The setup is similar to a plane, with two additional requirements:

- you need to setup your XPlane joystick to map the collective stick to flaps
- you need to map a key or joystick button to turn on and off the “generator1” electrical system

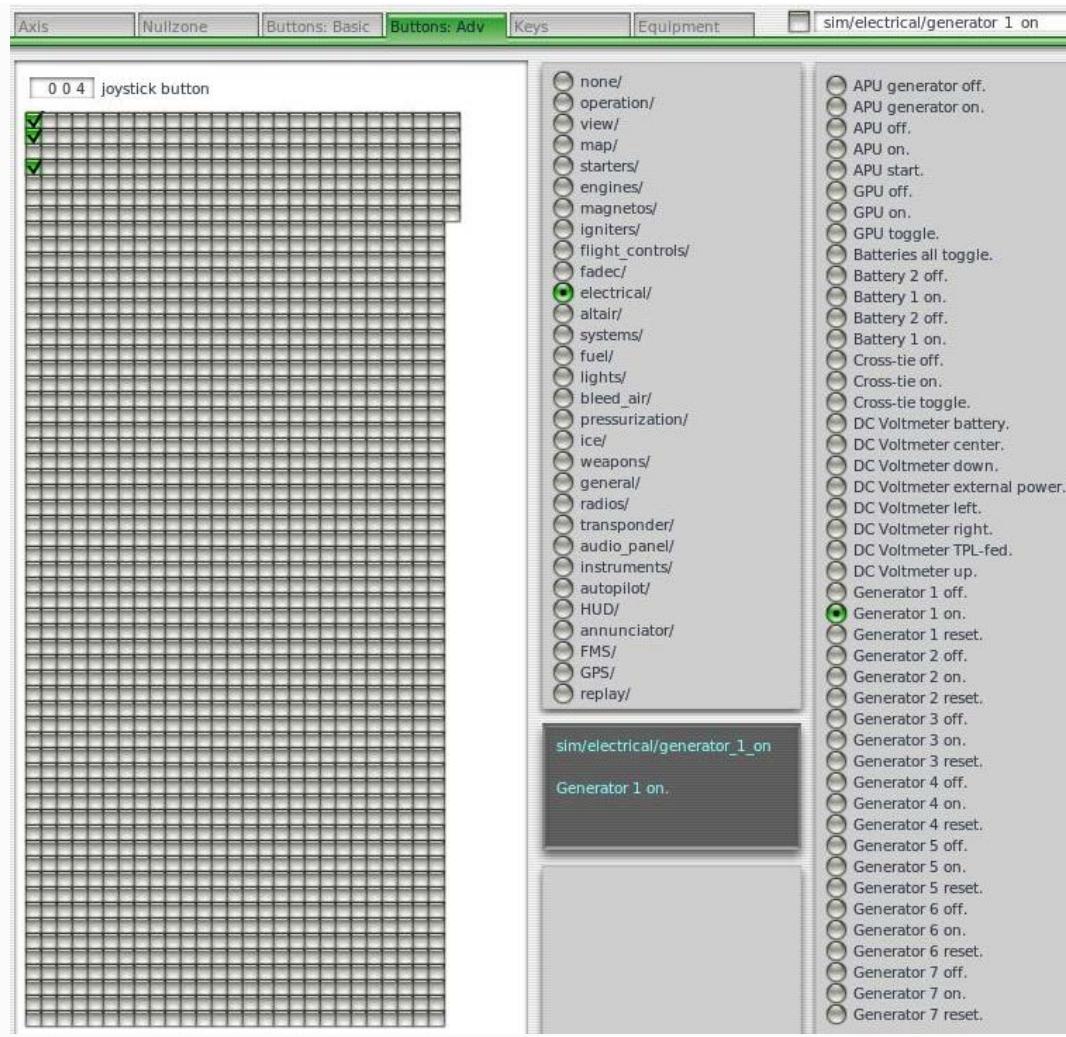
These strange requirements are because of limitations in the remote control of helicopters in X-Plane 10. The flaps input is something that ArduPilot SITL is able to read remotely while not interfering with flight of the helicopter. The “generator1 on/off” is used to simulate the interlock switch (channel 8) in ArduPilot helicopter support.

Note that for “generator on/off” you do need to map two separate events, one for on and one for off. If using a two position switch then map one to the switch on position and the other to the switch off position.

See this example for typical joystick setup



and this one for mapping the generator on/off switch to a joystick switch



A full set of parameters for the Bell JetRanger Helicopter in X-Plane 10 are available here

<http://uav.tridgell.net/XPlane/>

You also need to start SITL with the model set to "xplane-heli" instead of "xplane" to activate Helicopter controls.

The startup procedure for a helicopter is:

- set interlock on (so RC input channel 8 is low)
- set zero collective (so RC input channel 3 is low)
- arm the helicopter
- set interlock off (so RC input channel 8 is high)
- wait for the head to reach full speed
- takeoff

## HITL Simulators

Hardware In The Loop (HITL) simulation replaces the vehicle and the environment with a simulator (the Simulator has a high-fidelity aircraft dynamics model and environment model for wind, turbulence, etc.)

The physical AutoPilot hardware is configured exactly as for flight, and connects to your computer running the simulator (rather than the aircraft).

This section contains HITL simulation solutions.

**Note**

At time of writing X-Plane and Flightgear simulation for ArduPilot only support Plane (not Copter or Rover).

## X-Plane Hardware in the Loop Simulation

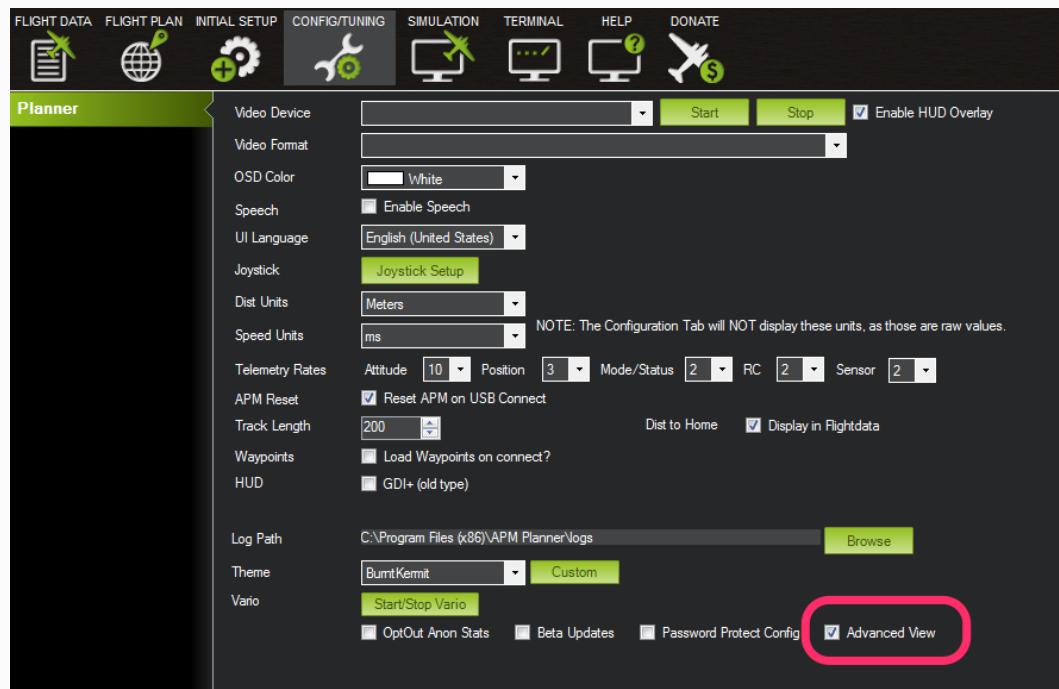
This article explains how to set up *X-Plane Hardware in the Loop Simulation*.

**Note**

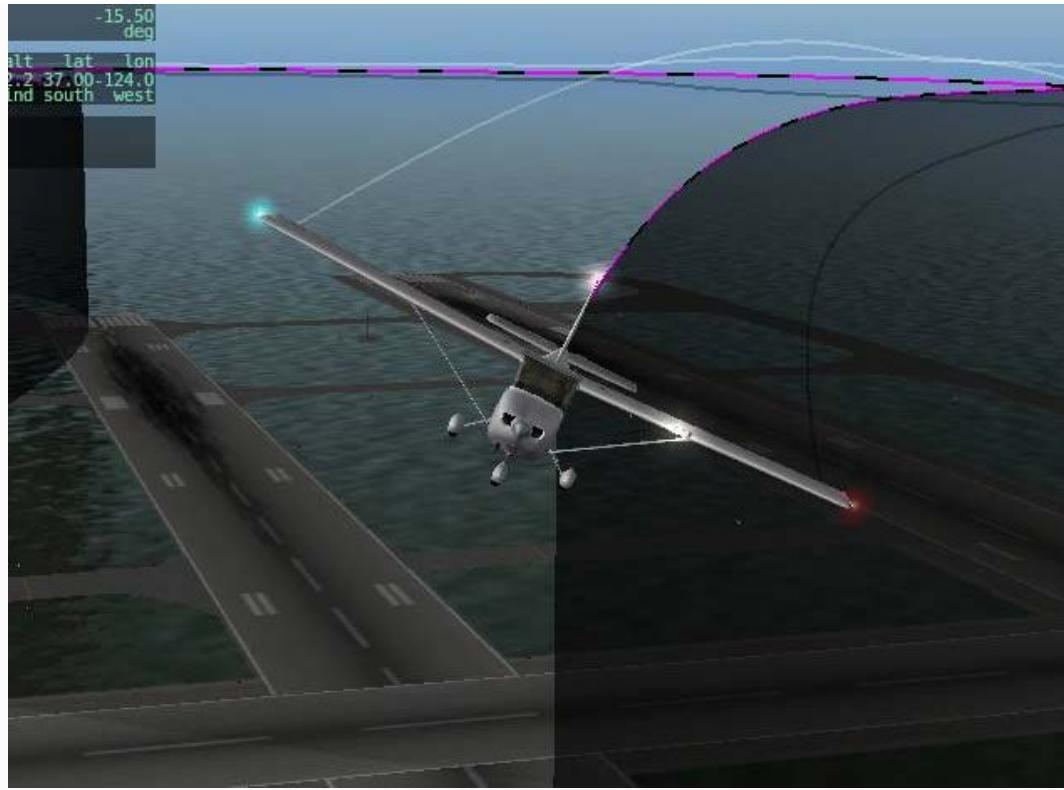
This simulator is Plane specific.

### Overview

The Simulation tab is visible on the top icon row when “Advanced View” is checked in the Config tab.

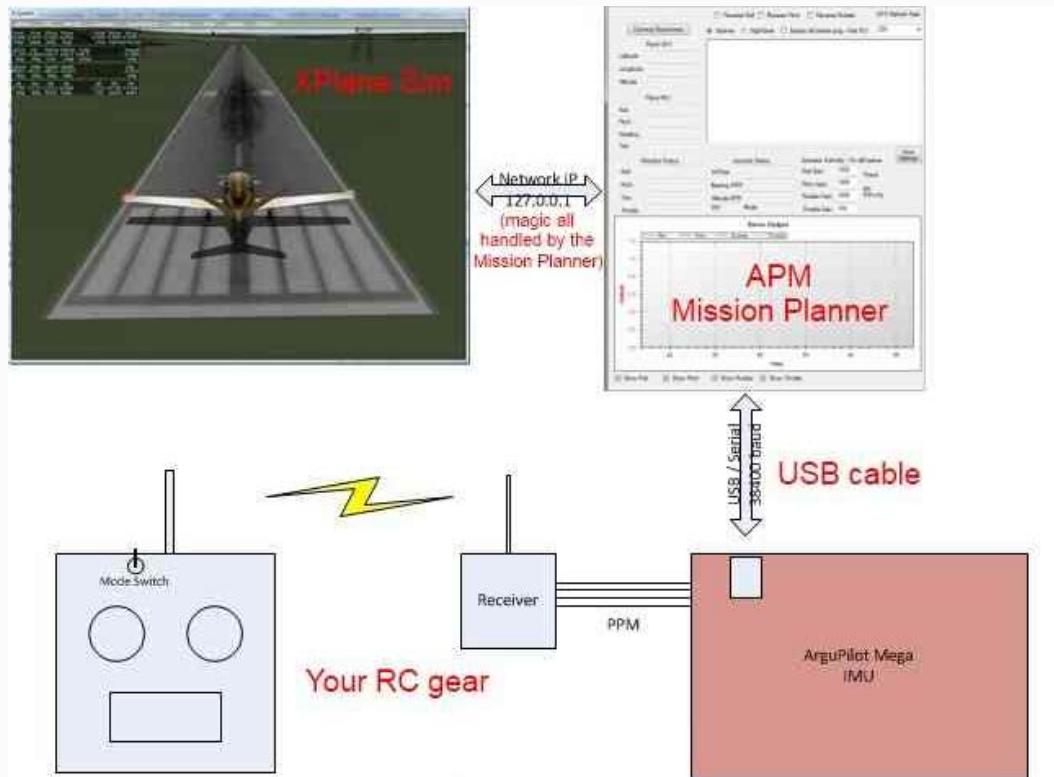


### How to set up a full hardware-in-the-loop simulation with X-Plane



It's easy to create a powerful "hardware in the loop" simulator with APM, with a Flight Simulator "fooling" APM with sensor and GPS data like it would experience in flight, and then APM flying the plane accordingly.

The overall setup looks like this:



## Setup

1. Download the [X-Plane 10 flight simulator](#), if you don't already have it. The demo is free but times out

after ten minutes. It's well worth buying the full version, which is just \$29.99. We support the 10.x versions.

2. Load the HIL Simulations version of the APM software with the Mission Planner (circled in red below; pick the one for the aircraft you want to simulate).
3. If you haven't done a setup already, press "APM Setup" on the Mission Planner Firmware page, which opens another dialog. Follow the instructions on the "Reset", "Radio Input" and "Mode" tabs of the dialog. Then you're ready to sim!



## Connecting APM and X-Plane

The Mission Planner will serve as your bridge to X-plane and will send the output from your APM's serial port connection to the flight simulator.

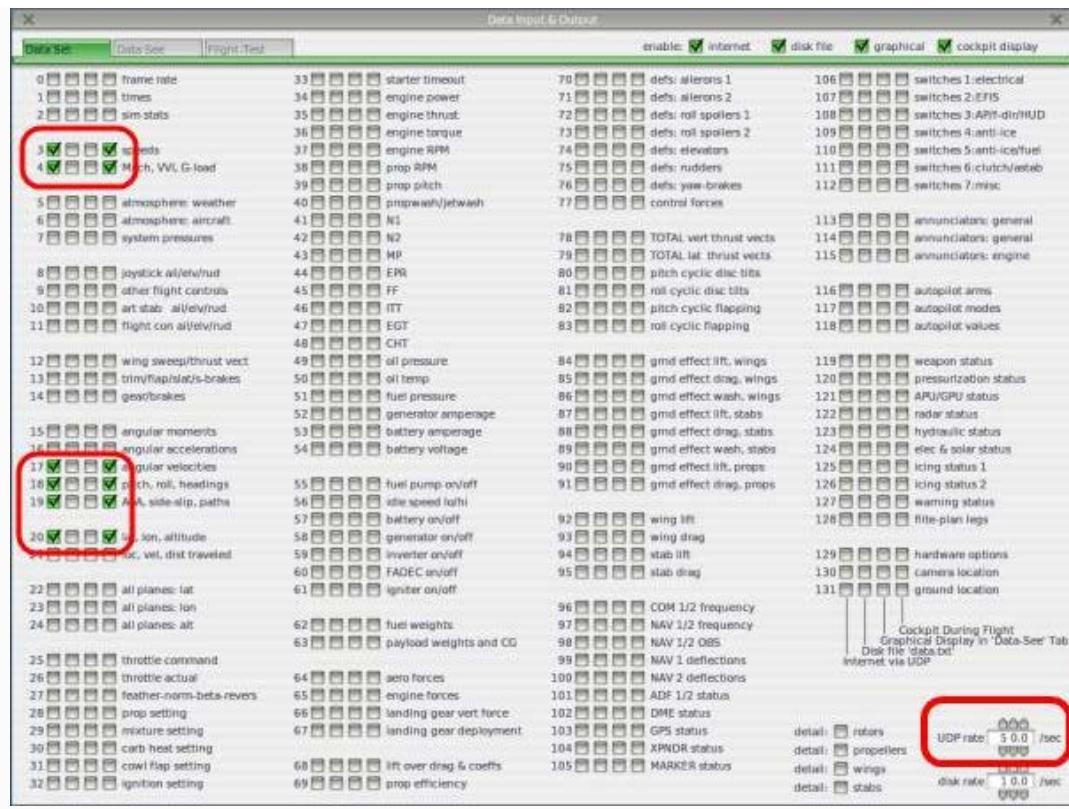
NOTE: If you're using a Mac, you can use a Perl-based solution (this also works on Windows, if you prefer Perl). Here's what you need: [X-Plane.pl](#)

### X-Plane Settings

X-Plane communicates over a network interface to the Flight Planner. Because of this you can run X-Plane and the Flight Planner on different computers. This is helpful if X-Plane running on a slower computer. Frame rate is very important.

You can choose any aircraft in X-Plane, but the most realistic one for our purposes is the PT-60 RC airplane. These are the screen shots that show the necessary settings you need to set in the Settings Menu – Data Input and Output:





You may also want to turn on the Frame Rate to display on the screen.

On the IP screen on Xplane, change the UDP data port to 49005 as shown below. (The Mission Planner also has an advanced IP setting dialog, but leave it at the default of 49000. 49005 is the UDP port coming in from Xplane, 49000 is the UDP port going out from the Mission Planner.)



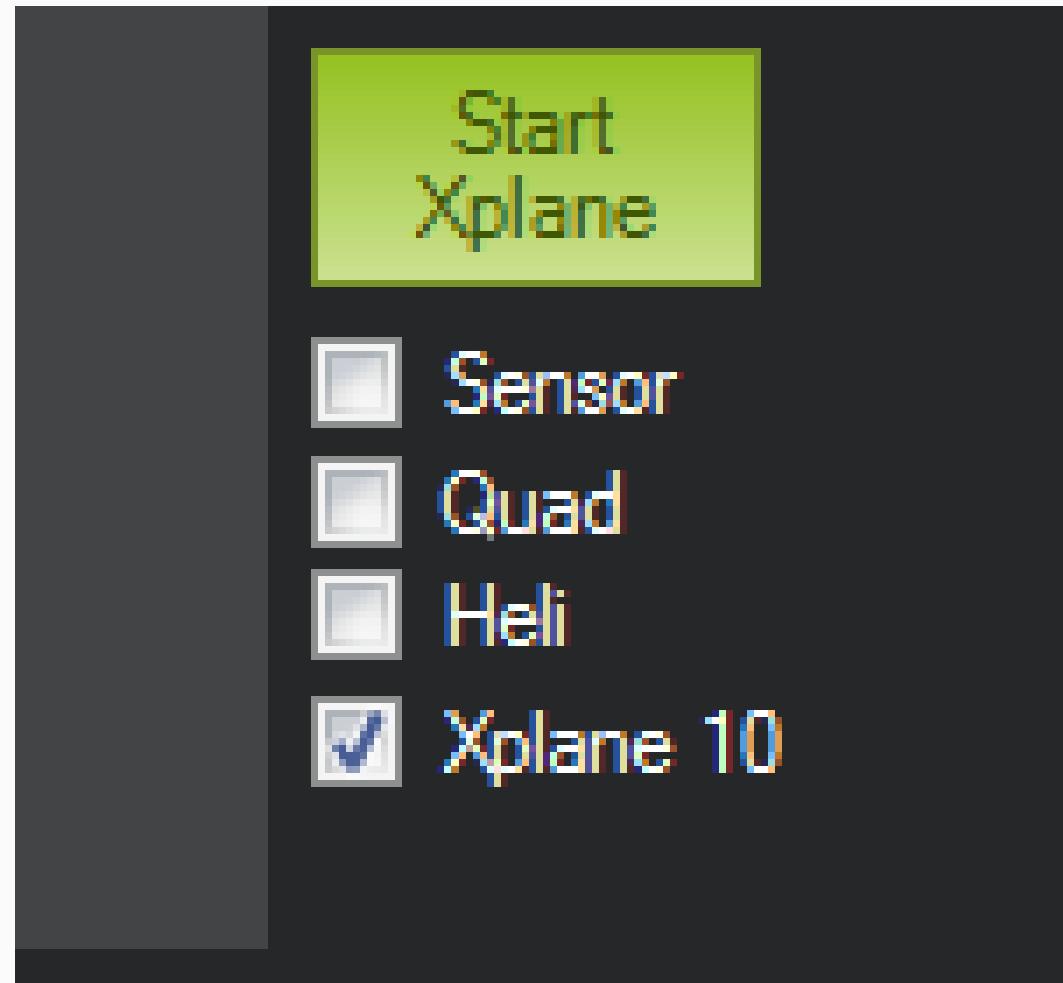
If you are running X-plane on the same computer as APM Mission Planner, then enter the loopback IP address (127.0.0.1) as shown above. If you are running X-plane on a different computer, enter the IP address of the other computer here(for example 192.168.1.1).

### Flight planner settings

Now you're ready to run the simulation. Start the Mission Planner, selecting the COM port APM is assigned to and 115200 baud as the speed. Click on the Simulation tab and you will be taken to this screen:



If you are using X-Plane 10.x, you must check the "XPlane 10" checkbox shown here:



If you are running APM Mission Planner on the same computer as X-Plane, then the default IP settings should be fine. But if you're in doubt, you can check the "Advanced IP Settings". The IP address should be 127.0.0.1 and the port should be 49000, as shown. If you are running X-plane on a different computer, though, enter the IP address of the other computer here (for example 192.168.1.2). The port should be 49000 in either case.

Once you've selected Sim Link Start/Stop, if APM is connected and Xplane is running, you should see the Mission Planner Sim screen fill with data.

Turn your transmitter on and check your link to your receiver. (Remember that if you're using APM 1 you must plug an ESC and LiPo into the RC pins of APM to power the RC receiver; this is not necessary on APM 2, where the radio is powered by the USB). When you move the sticks you should see the APM outputs change on the Flight Planner Simulator window.

For first testing, set your three RC toggle switch modes to Manual, Stabilize and RTL. You can do that via the Mission Planner's MAVLink setup. Hit control-G in the Mission Planner once you're connect to go to the setup screen, where you will find the Modes setup.

### **Flight test**

Now fly the plane in the sim with your RC transmitter. First, hit the "C" key so you can see your plane up close and from the side. Try each stick on your RC transmitter and make sure that the control surfaces go in the right direction in manual mode. If they don't, reverse them at your RC transmitter as you would with a normal RC airplane.

You can now try a flight. Reload the aircraft in Xplane, hit "Shift-8" so you can see it from behind, and hit "b" to release the brakes. The aircraft should begin moving forward. You can steer it on the ground with your rudder stick. Once it's picked up speed, give it a little up elevator to take off. You should be able to fly it like a RC flight simulator.

If it's flying well, try moving your toggle switch into stabilize mode. The aircraft should immediately start flying flat and level. If it rolls upside down and wants to fly that way, you'll need to reverse the ailerons on the SIM model on your transmitter. Likewise for diving.

If stabilization works well, try RTL. The aircraft should return to the start of the runway and circle there at 100m.

Once that is working, trying entering some waypoints and flying an auto mission. In the Mission Planner read the stored waypoints and let it reset home to the stored position (which will now be the airport in your Xplane simulator that you're flying from). Click to add a few waypoints and switch into auto to watch the show!

In general, the correct order to load and run everything is as follows:

1. Use the Mission Planner to load the Simulator version of the APM code and perform the setup.
2. Start Xplane. After it is initialized and while the plane is sitting on the ground hit the "a" key for a rear view and then the "p" key to pause.
3. Start the Mission Planner, select the right port and baud for your APM board and connect to APM.
4. In the Mission Planner simulation tab, click the button in ArduPilotSim to connect. You should see values appear in the output fields. Switch your TX mode switch to the manual position and verify that the outputs are moving with your TX sticks.
5. Make sure your throttle stick is down.
6. Switch to Xplane. You are ready to go. Click the "p" key to unpause. Hit the "b" key to unlock the brakes. Advance the throttle and take off!

### **Notes**

- I have been using the stock PT-60 in Xplane with good results. I would recommend a cruise airspeed of 15 m/s
- One quirk of the PT-60 is that it bounces around a lot on the ground and if you are not careful it has prop-strikes followed by simulated engine failure, which is really annoying. If the engine has stopped, reload the plane. Hold the pitch stick for a modest climb and go to full throttle.
- Remember that APM always sets its “home” location automatically at the field, which in this case is whichever airport you’ve set Xplane to start at. It will overwrite any home you may have set in the Mission Planner. If you want the flight sim to start at a different airport, you must select that from Xplane’s “Location” menu. You can only select the provided airports, not just any place in the world you want to fly.
- If you want to play with the code and load it via Arduino,

, rather than the pre-compiled hex file in the Mission Planner, you can. You just need to change your APM\_config.h file to the following:

```
#define HIL_MODE          HIL_MODE_ATTITUDE
```

HIL\_MODE\_SENSORS is not currently (Jan 2012) working. HIL\_MODE\_ATTITUDE informs Plane of the attitude of the plane but not the accelerations or roll rates. HIL\_MODE\_SENSORS originally was going to inject the lower level physics into the real sensor code, from which would (hopefully) be derived the same or similar attitude figures, plus more information from the Barometer and so on.

Software-in-the-loop (SITL) sim has taken this further, emulating !Arduino registers and injecting the sensor information at the lowest level possible. It is therefore the better option if you want to sim with the most realistic inputs.

### Error messages and fixes

1. Can’t open serial port : The com port has not been selected or the port is in use. Do you have a terminal open to the APM? Note that the mission planner loads the connection window coms port menu with every port it finds. If the USB driver failed to hook up and did not promote a port into !Windows, then this menu will be missing the actual port or may default to another port.
2. Socket (IP) setup problem. Do you have this open already? : You have another program running that is using the IP Port and conflicting with the data communications between Xplanes and APM Planner.
3. Xplane Data Problem - You need DATA IN/OUT 3, 18, 19, 20 : Please redo the setup and make sure all the boxes are ticked. You may need to restart X-Plane.
4. Bad Gains!!! : One of the simulator gain numbers is invalid.
5. NO SIM data : ArduPilot Mega Planner is not receiving any data from Xplane. Please check your Xplane settings.
6. The radio has no control—you push the sticks and nothing happens. Are your PID setting zero? Have you setup up your Radio inputs? The configuration defaults may not be appropriate for your radio. Please review the minimum, maximum and trim figures and if necessary use the radio Setup mode either from the mission planner or from the command line setup mode.
7. Flight controls are way off. You need to push the sticks way over to fly the plane. Check your Radio setup in the APM. Did you configure the Radio? See (7).

### Debug

1. The on-screen status details only update after a valid connection to the APM has been made.
2. Open Terminal, and verify what you see. If the text is readable go to the second point below, if it is not look at the first point
  1. When you click Connect you should see the APM header printed in the text box. If you don’t you may not have the correct Comport or Baudrate
  2. Verify the setup by opening the Terminal in APM Planner and see what it prints out. it should start

with the APM header and continually spit out AAA???? forever. If you don't see the AAA's then you may have uploaded the wrong firmware (not simulation mode) or if you're loading the code manually, not made the necessary changes to the **APM\_Config.h** file.

3. Try re-doing the radio calibration after selecting the simulation, and make sure the throttle gain is set to 10,000 in the simulation tab.

## Python Scripting with XPlane

This section shows how to use Python for some basic scripted acrobatic moves for fixed-wing airplanes (including a left roll, a right roll and most of a loop). The instructions come from the tutorial [Scripted Fixed-Wing Acrobatics](#).

1. You need a python based script. You can start with the one I've put together. It's attached here. ["simple\\_script.txt"](#)
2. With X-plane, the APM Mission Planner (MP) and the script open, get your plane in the air and press "p" for pause. "a" will put you in chase view ("w" to go back in the plane) and "-" and "=" will zoom your view of the plane in and out.
3. In the MP click on the "Flight Data" button, then select the "Actions" tab underneath the Heads-Up-Display.
4. Click on the "Script" button. This will bring up another window with a preloaded script (for Copter). Switch to the already open "simple\_script.txt" and select-all then copy.
5. Go to the python script window and paste "simple\_script.txt" in the python script window. DO NOT close it yet.
6. Go to X-plane, and press "p" again to un-pause the plane.
7. Go back to your python script window and close it. This will trigger the execution of the script when you click "yes" in the pop-up.
8. Watch your plane do some rolls, and most of a loop.

## FlightGear Hardware-in-the-Loop Simulation

[FlightGear](#) is an open-source free flight simulator. This article explains how to use this hardware-in-the-loop simulator with Plane (only).

Note

This is a community-submitted article. It should work, but has not been verified by the official dev team. Questions and requests for help should be made in the DIY Drones forums.



**Flightgear screenshot**

## Things you need

1. [Flightgear](#) (tested Flight Gear v2.x, currently at v3.4)
2. [Mission Planner](#) (Windows)
3. ArduPilot HIL on your APM (via APM Planner, currently (April 2013) only v2.68 HIL works, see below)
4. [Mavlink.xml](#) data format file for flightgear (Right Click and Save as)
5. Recommended for Windows OS: the file: [system.fgfsr](#)c placed into the “C:\Program Files (x86)\FlightGear\data” folder. This will load the Rascal RC plane and a few other parameters. More information on what is loaded [here](#), you can alter it to your own requirements.

## How to make it work

### Warning

Safety first: Disconnect the 3 wires from your brushless motor otherwise it will start when you advance the throttle (if in manual mode).

1. Download and install the FlightGear Flight Sim if you have not already
2. Copy Mavlink.xml to C:\Program Files (x86)\FlightGear\data\Protocol
3. In the Mission Planner > ‘Configuration tab | Adv Parameter List’ save your current parameters so you can reload them after the simulation
4. In the Firmware tab of the MP, choose ‘Pick previous firmware’, from the drop down list pick the 4th one down, firmware should show as Plane v2.68, then click ‘HIL Simulator | Plane’ to load the Plane firmware onto your APM.
5. Connect to the APM, then open the Simulation tab of the Mission Planner
6. Check the FlightGear radio button, then click the ‘Start FG Plane’ button.
7. After FG has finished loading click the ‘Sim Link Start/Stop’ button in the MP
8. In FG press ‘v’ on the keyboard to get an outside view of the plane, ‘x’ to zoom, move your aileron stick to see if the planes ailerons move, great if they do, go through this list again if not.
9. Now in the Mission Planner do your usual radio setup including calibration and mode switch setup, then you can fly Missions and test the APM just like you would on a real plane.

### Note

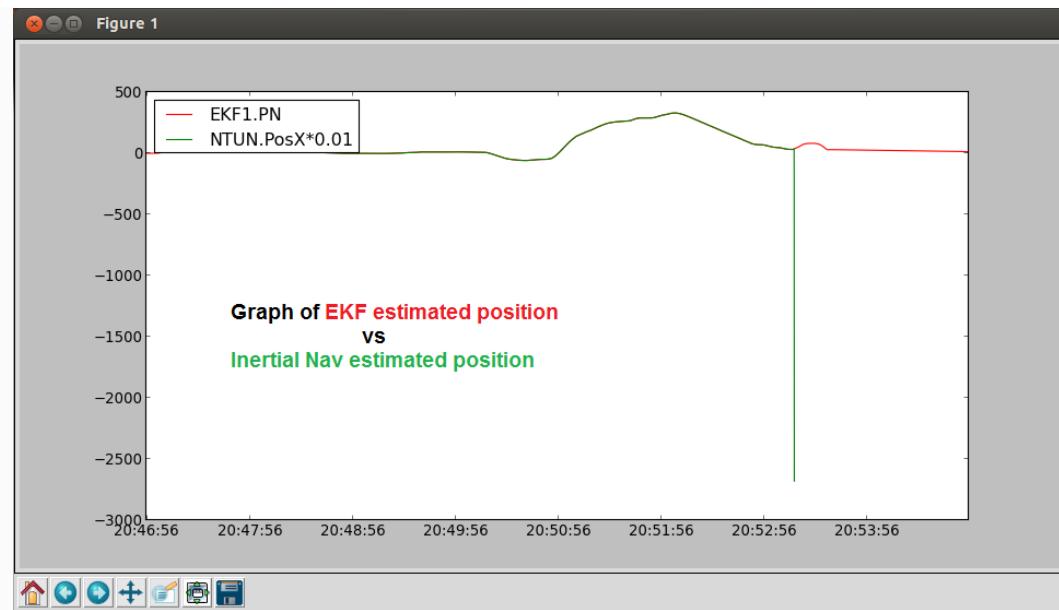
If you used the file “[system.fgfsr](#)c” then you can just increase your throttle to fly the Rascal plane. If you didn’t and are flying the default Cessna, hold down ‘s’ on the keyboard to start the plane’s engine, then advance your throttle to take off.

## Testing with Replay

### Introduction

Replay is a program that takes a dataflash log file and replays it through the latest master code allowing a sort of simulation based on real data. This can be useful when trying to recreate the exact situation which produces a bug or to test EKF tuning parameters to see how they would have performed in the same situation. Replay only runs on Linux/Ubuntu and only using dataflash logs from a high speed CPU such as the PX4/Pixhawk running a version of Copter/Plane/Rover from May 2014 or later (i.e. AC3.2-dev or higher).

It is recommended that if a problem is reproducible that the dataflash log be generated with both the *LOG\_REPLAY* and *LOG\_DISARMED* parameters set to 1.



## Dataflash log messages required for Replay

If `LOG_REPLAY` is not set, the following dataflash messages must be enabled: AHRS2, BARO, EKF1, EKF2, EKF3, EKF4, GPS, IMU, IMU2, MAG, MAG2.

## Building Replay

On your Linux or Ubuntu machine, from the root directory of an ArduPilot repository:

```
./waf configure --board=linux
./waf build --target=tools/Replay
```

### Note

You may need to install `sudo apt-get install pkg-config`  
This will create a file called `build/linux/tools/Replay`.

## Using Replay

Display the Replay help instructions:

```
build/linux/tools/Replay -- --help
```

Run a log through Replay to generate the plot and EKF data files:

```
build/linux/tools/Replay -- MyLogFile.BIN
```

### Note

You may need to explicitly set the loop rate with `-r400`  
This will produce six output files: `plot.dat`, `plot2.dat`, `EKF1.dat`, `EKF2.dat`, `EKF3.dat`, `EKF4.dat`

Look at the raw data to see which values are available to be plotted:

```
less plot.dat (you can replace plot.dat with any of the other six files produced)
```

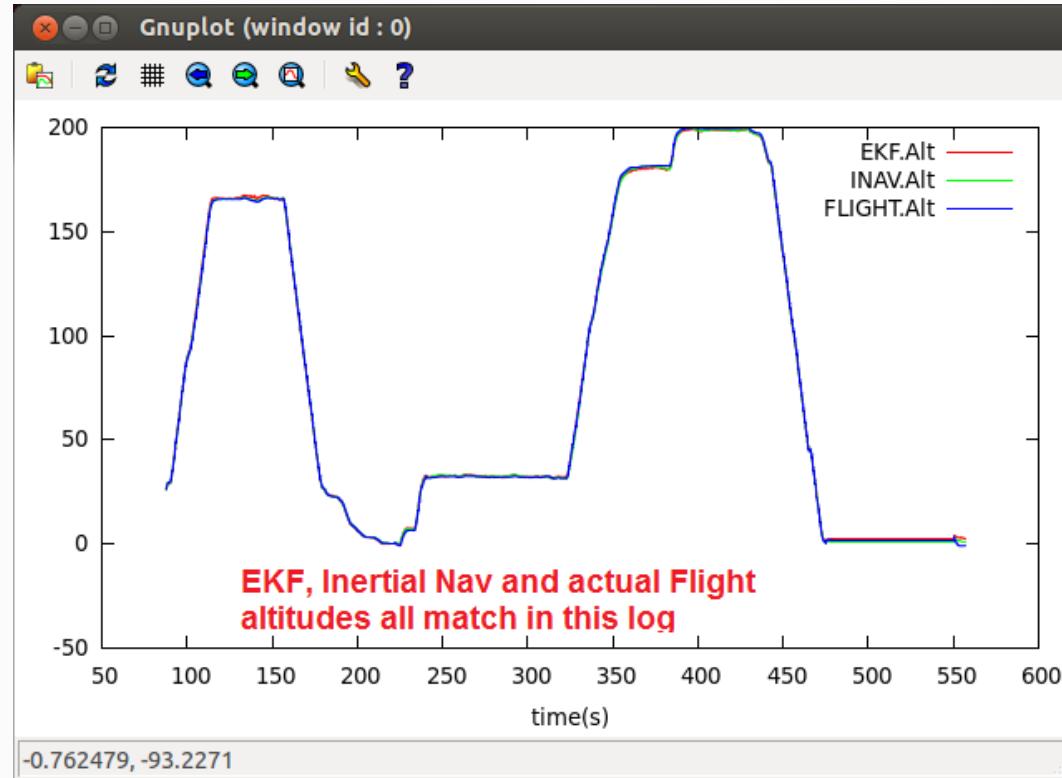
```

time SIM.Roll SIM.Pitch SIM.Yaw BAR.Alt FLIGHT.Roll FLIGHT.Pitch FLIGHT.Yaw FLIG
HT.dn FLIGHT.dE FLIGHT.Alt DCM.Roll DCM.Pitch DCM.Yaw EKF.Roll EKF.Pitch EKF.Yaw
INA.V.dn INAV.dE INAV.Alt EKF.dN EKF.dE EKF.Alt
15.100 0.0 0.0 0.0 100.11 3.1 1.1 00.0 0.00 0.00 0.03 3.1 1.0 97.0 1.5 0.
7 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.43
45.581 0.0 0.0 0.0 -193.44 -3.4 -1.4 83.0 0.00 0.00 0.03 -3.4 -1.9 97.1 -4.5 -3.
7 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.42
45.681 0.0 0.0 0.0 -193.44 -3.4 -1.4 83.0 0.00 0.00 0.03 -3.4 -1.8 97.1 -4.5 -3.
7 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.40
45.780 0.0 0.0 0.0 -193.44 -3.4 -1.5 83.0 0.00 0.00 0.02 -3.4 -1.8 97.1 -4.5 -3.
7 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.39
45.880 0.0 0.0 0.0 -193.44 -3.4 -1.5 83.0 0.00 0.00 0.02 -3.4 -1.8 97.1 -4.4 -3.
7 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.37
45.980 0.0 0.0 0.0 -193.44 -3.4 -1.5 83.0 0.00 0.00 0.02 -3.4 -1.8 97.1 -4.4 -3.
6 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.35
46.080 0.0 0.0 0.0 -193.44 -3.4 -1.5 83.0 0.00 0.00 0.02 -3.4 -1.8 97.1 -4.4 -3.
6 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.32
46.181 0.0 0.0 0.0 -193.44 -3.4 -1.5 83.0 0.00 0.00 0.01 -3.5 -1.8 97.1 -4.4 -3.
5 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.29
46.280 0.0 0.0 0.0 -193.44 -3.4 -1.5 83.0 0.00 0.00 0.01 -3.5 -1.8 97.1 -4.4 -3.
4 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.26
46.381 0.0 0.0 0.0 -193.44 -3.4 -1.5 83.0 0.00 0.00 0.00 -3.5 -1.8 97.1 -4.3 -3.
3 100.3 0.00 0.00 0.00 1000000.00 -1000000.00 -193.23
plot.dat

```

Use the simple plotit.sh script to graph some data. Below is the command to compare the EKF calculated altitude with the older Inertial Nav calculated altitude and the flight's actual altitude.

```
./Tools/Replay/plotit.sh EKF.Alt INAV.Alt FLIGHT.Alt
```



Use the more complex mavgraph.py to graph the data

```
mavgraph.py MyLogFile.BIN EKF1.PN NTUN.PosX*0.01
```

This example compares the EKF estimated North-South position from home vs the older Inertial Nav estimated position. See image at the top of this page for the resulting graph.

## Changing parameters

Simulation parameters may be changed before replaying a log using the option: `-pNAME=VALUE` (this sets the parameter `NAME` to `VALUE`). The parameters which may be edited are those listed by running the `:ref:`param show`` command in `SITL <setting-up-sitl-on-linux>`.

For example, to change the EKF velocity delay parameter from 220ms to 400ms, run the command:

```
./build/linux/tools/Replay -- -pEKF_VEL_DELAY=400 MyLogFile.Bin
```

## The ArduPilot Autotest Framework

ArduPilot has an automatic testing framework based on SITL. The autotest framework is what produces the web pages at <http://autotest.ardupilot.org>

You can also manually run the autotester from the command line. That is useful when you want to add features to the autotest or want to test some new flight code on your own machine with auto scripting.

To use the autotester you need to first [get SITL running](#). After that you should run:

```
cd ardupilot
./Tools/autotest/autotest.py --help
```

that will show you the command line options for the autotester. If you get any python errors it probably means you are missing some required packages. Go and check on the SITL setup page and see if you are missing anything.

### Test actions

The autotester supports a long list of possible test scripts. If you run:

```
./Tools/scripts/autotest.py --list
```

you will see what test scripts you can run. You can then add those commands on the command line to run them. For example, to build the fixed wing code and then run a test flight do this:

```
./Tools/scripts/autotest/autotest.py build.Plane fly.Plane
```

the results (and log files) will be put in the `../buildlogs` directory.

You can also ask it to display a map while it is flying, which can make watching autotest a bit less boring! Run it like this:

```
./Tools/scripts/autotest/autotest.py build.Plane fly.Plane --map
```

you will actually see the map appear twice, once for when it loads the default parameters, and then for the real flight. Just close the first one.

### Changing the test scripts

Each of the tests is in `Tools/autotest`. For the fixed wing tests look at `Tools/autotest/arduplane.py`.

## Debugging

This section is for topics related to debugging ArduPilot source code.

### Using MAVExplorer for log analysis

MAVExplorer is a log exploration tool. It is based on the same concepts as mavgraph, but is interactive, and supports a community contribution model for pre-defined graphs.

#### Installing MAVExplorer

You will need the latest version of pymavlink and mavproxy installed. On Linux do this:

```
sudo apt-get install python-matplotlib python-serial python-wxgtk2.8 python-lxml  
sudo apt-get install python-scipy python-opencv python-pip python-pexpect python-tk  
sudo pip install --upgrade pymavlink mavproxy
```

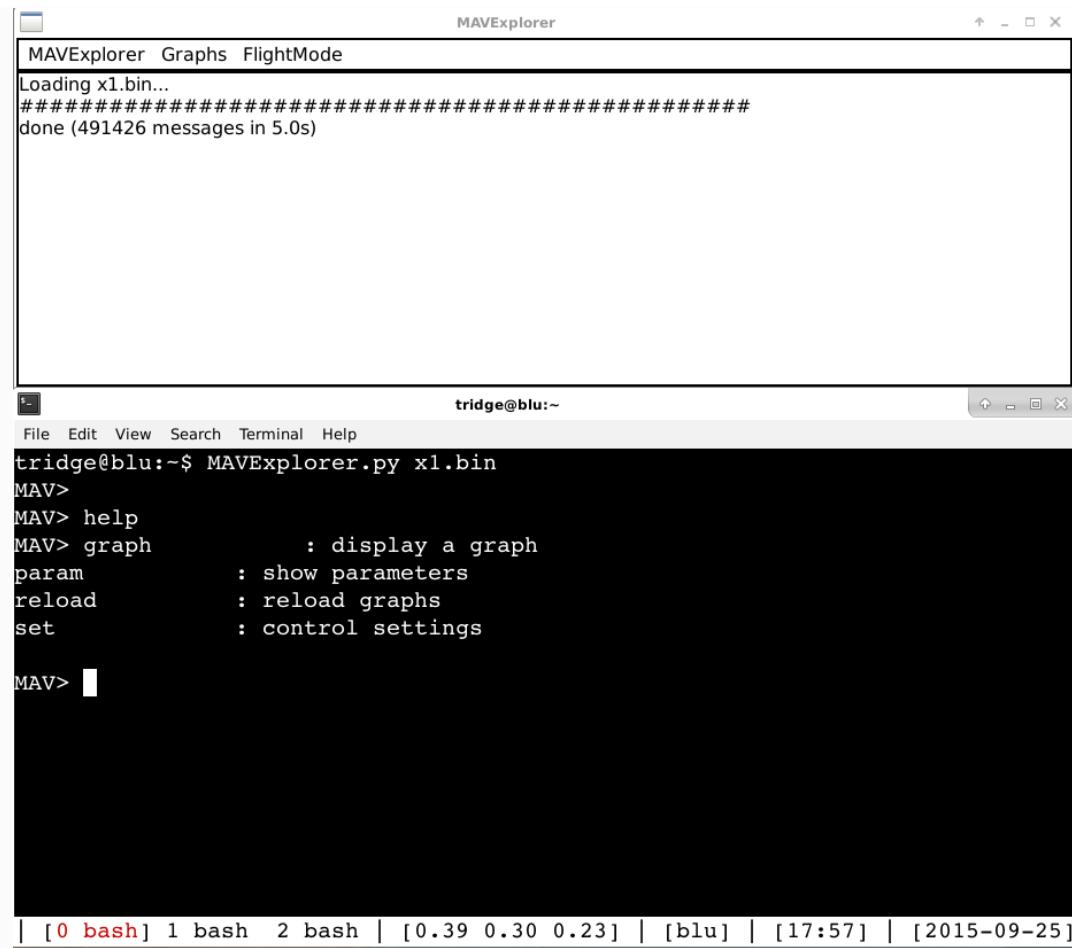
after running that you should have a new command "MAVExplorer.py" in your path.

#### Starting MAVExplorer

To start MAVExplorer just run it with a filename as an argument:

```
MAVExplorer.py mydata.bin
```

it supports MAVLink telemetry logs, or DataFlash Logs. After starting it you will end up with two windows, like this:

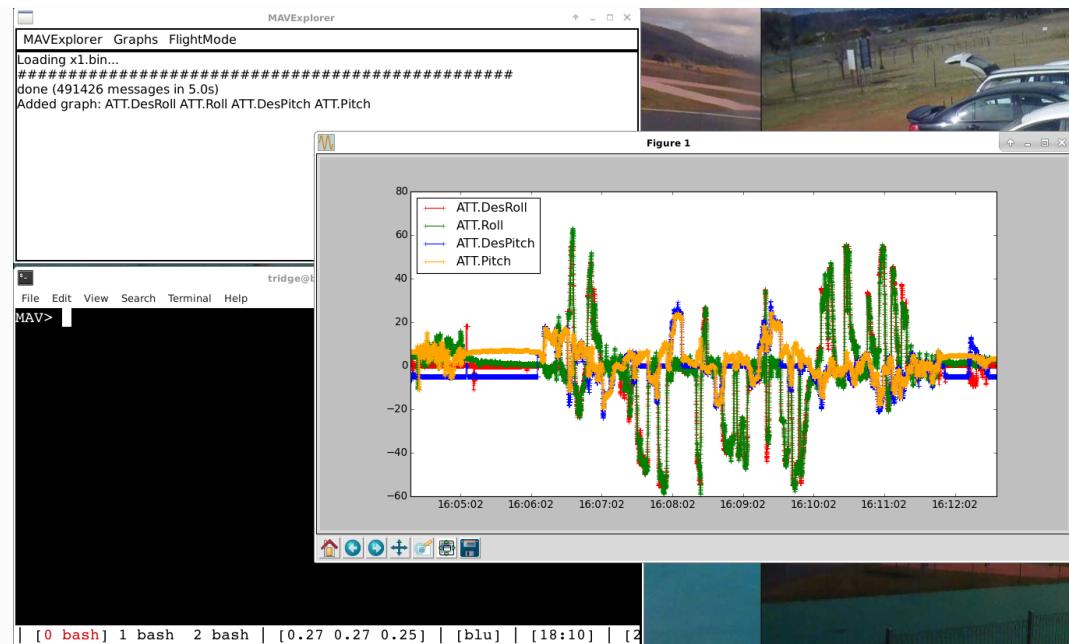


The top window is the “console” and has your menus and the status messages. The bottom window is your terminal, and has the “MAV>” prompt for typing commands.

### Using pre-defined graphs

MAVExplorer comes with an extensive set of pre-defined graphs. Those graphs appear in the Graphs menu in the console. Only graphs that are relevant for the log you are viewing will appear in the menus.

To display a graph just choose it in the Graphs menu. You can display several graphs at once if you want to.



## Graphing Manually

You can also graph any data from the log using the graph command in the terminal at the MAV> prompt. Just type graph followed by the expression. For example:

```
graph ATT.Roll ATT.Pitch
```

to help you create graphs quickly you can use the TAB key to complete and list available messages and fields. For example, if you did this:

```
graph <TAB><TAB>
```

you would see something like this:

The figure shows a terminal window with the text:

```
tridge@blu:~
```

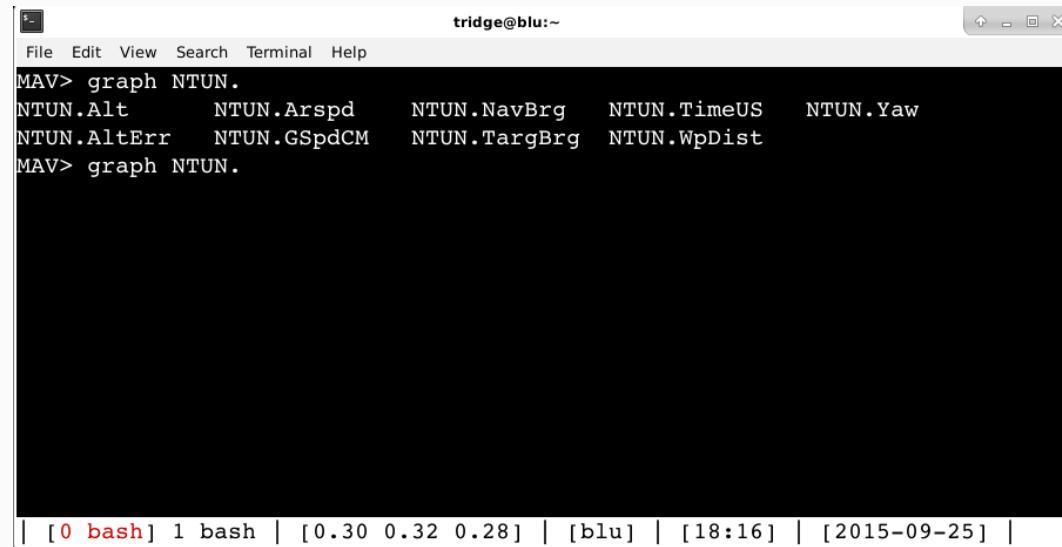
The terminal prompt is MAV>. The user has typed "graph" and is using the TAB key to complete the message name. The list of available messages and fields is displayed:

	AHR2	BARO	EKF1	ESC3	IMT	MAG	MSG	PIDP	PM	RCIN	STAT	VIBE
ARM	CMD	EKF2	FMT	IMT2	MAG2	NTUN	PIDR	POS	RCOU	STRT		
ARSP	CTUN	EKF3	GPA	IMU	MAV	ORGN	PIDS	POWR	RFND	TECS		
ATT	CURR	EKF4	GPS	IMU2	MODE	PARM	PIDY	RAD	SONR	TERR		

At the bottom of the terminal window, the timestamp is shown as [2015-09-2].

the list of possible message names has been listed. If you type part of the message name you want then you can use <TAB> to complete the name.

After the message name you need to type a '.' followed by a field name. Again you can TAB complete, for example:



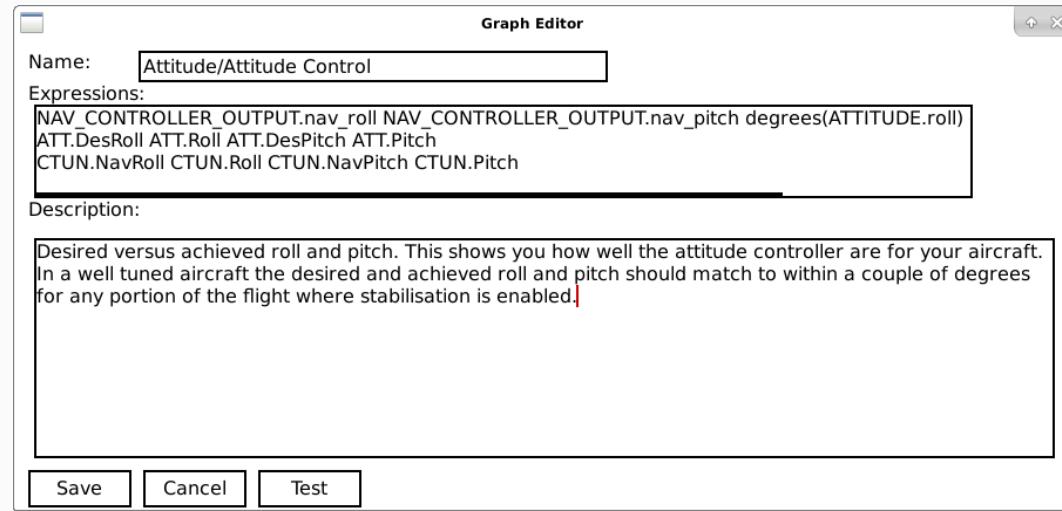
```
tridge@blu:~  
File Edit View Search Terminal Help  
MAV> graph NTUN.  
NTUN.Alt      NTUN.Arspd     NTUN.NavBrg   NTUN.TimeUS   NTUN.Yaw  
NTUN.AltErr   NTUN.GSpdCM   NTUN.TargBrg  NTUN.WpDist  
MAV> graph NTUN.  
  
| [0 bash] 1 bash | [0.30 0.32 0.28] | [blu] | [18:16] | [2015-09-25] |
```

You can do this for multiple fields in one graph, allowing you to construct complex graphs quickly.

### Modifying pre-defined graphs

Whenever you use a pre-defined graph the graph expression gets added to your command line history. So you can just hit enter to refresh the history then up arrow to bring up the graph expression of the pre-defined graph you just displayed. You can then add new fields or editing existing fields and hit enter to display the new graph.

You can also save any graph you have just displayed using the “Save Graph” menu item under the “MAVExplorer” menu:



You can edit the name and description of the graph then press “Save” and your graph will be added to your set of pre-defined graphs. You can also use the “Test” button to test a graph before saving.

Note that the / separators in the name of the graph control how the graph will appear in the Graphs menu tree. So for example if you save a graph with a name of “Copter/Analysis/WPNav Analysis” then a “WPNav Analysis” menu item will be added to the Copter->Analysis submenu, automatically creating the submenus as needed.

Also note that each graph expression should be on a line by itself. If you have multiple lines for the expressions then they will be considered as alternative expressions (to cope with different types of logs).

## Graph Expressions

Graph expressions are arbitrary python expressions. You can use any functions from the python maths library, plus any functions from the mavextra module in pymavlink or your own mavextra module.

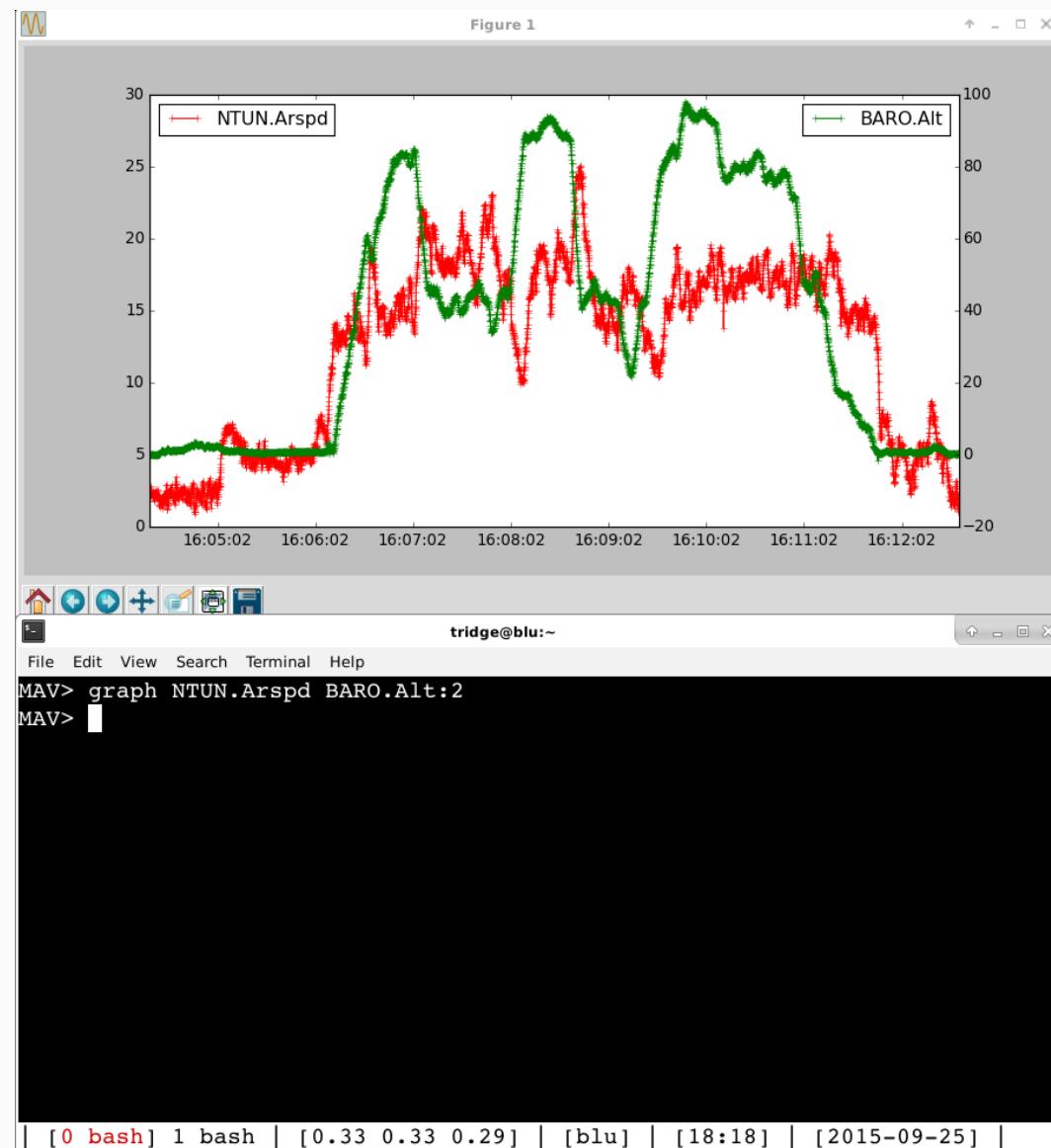
This allows you to easily graph mathematical expressions combining any variables in the log. For example:

```
graph sqrt(MAG.MagX**2+MAG.MagY**2+MAG.MagZ**2)
```

that will graph the total magnetic field strength (the length of the compass vector).

## Right and Left scales

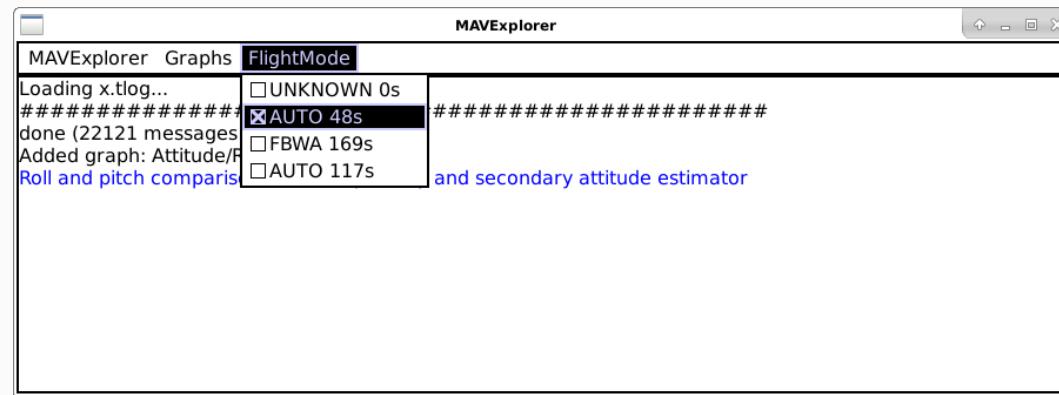
The default is that graphed values are shown on the left hand axis scale, all scaled together. To plane a field on the right hand scale just use ":2" on the end of the field name. For example:



## Selecting by Flight Mode

It is often useful to select only a part of a flight based on the flight mode. To do that use the FlightMode

menu:



The FlightMode menu will show all the flight mode changes in your flight, along with how many seconds it was in that mode. You can select which parts of the flight to include in subsequent graphs by selecting the appropriate flight menu items. If none are selected then the whole log is graphed.

### **Adding Conditions**

You can restrict graphs based on conditions expressed as python expressions of the available log variables. For example, if you wanted to only graph where the GPS speed was above 4 meters/second in a DataFlash log you could do this:

```
condition GPS.Spd>4
```

to clear the condition set it to the empty string

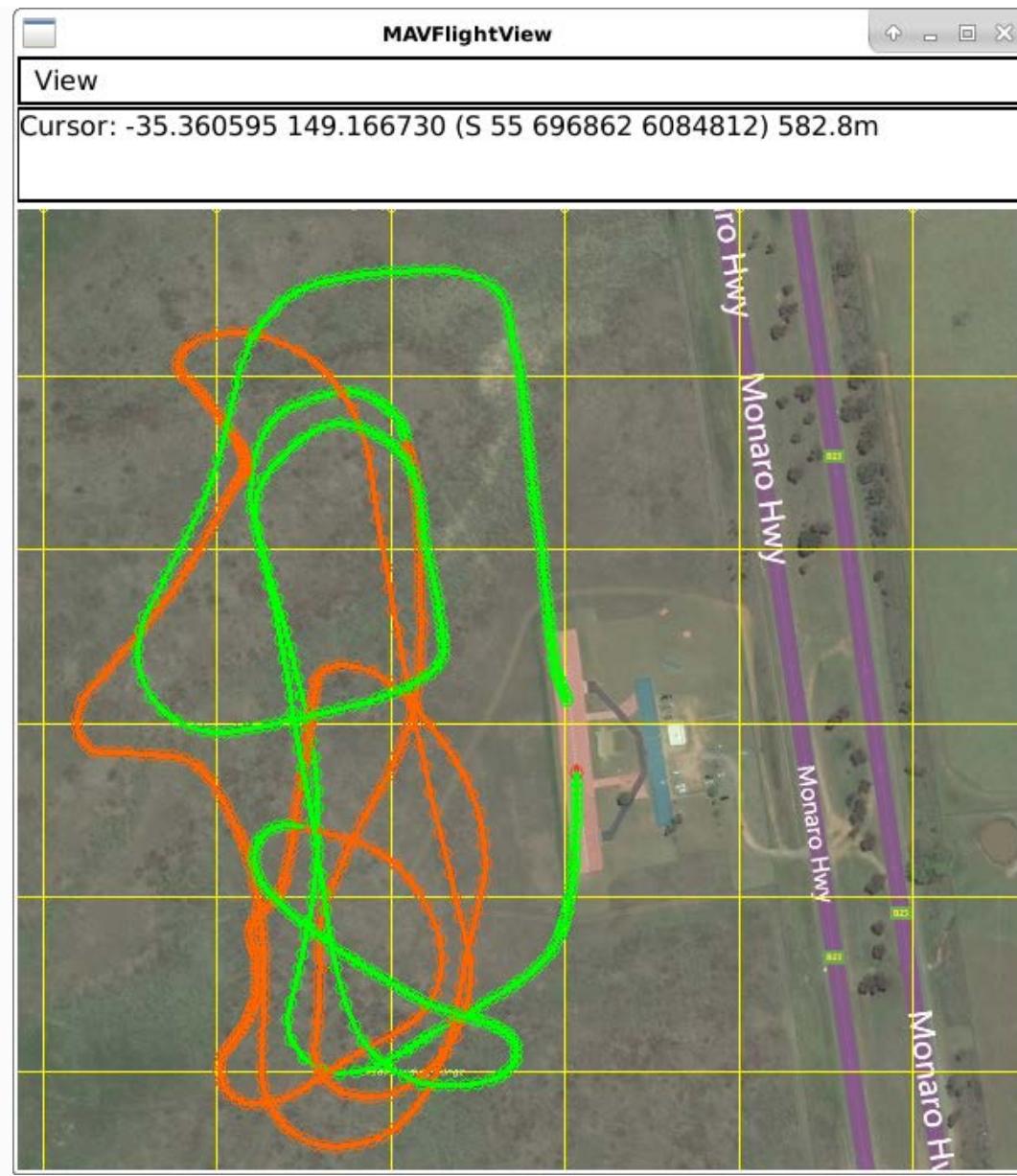
```
condition ''
```

### **Displaying a Map**

You can display a map showing your flight path using the map command:

```
map
```

A map will pop up following the current condition, like this:



You can zoom in and out using the mouse and the scroll wheel or + and - keys. You can measure distances using two left clicks. Colours are used to indicate flight modes.

You can optionally supply message types to use for the map. Any message type that has latitude and longitude elements can be used. For example, to show a map from both the GPS and POS messages in an ArduPilot log file you can do:

[map](#) [GPS](#) [POS](#)

The multiple tracks will be shown with sequentially darker colours.

### Graph Definition XML files

You will probably find it useful to add your own pre-defined graph definitions for commonly used graphs. These pre-defined graphs are created in XML files, and can be shared with other users of MAVExplorer.

The quickest way to create these graphs is to use the “Save Graph” feature, but you can also create the XML files manually using your favourite text editor.

You can see an example of the XML format here:

<https://raw.githubusercontent.com/ArduPilot/MAVProxy/master/MAVProxy/tools/graphs/mavgraphs.xml>

Create an XML graph file

MAVEexplorer looks in 3 places for XML files to get graph definitions from:

- in the current directory it looks for a file called "mavgraphs.xml"
- in your home directory it looks for any XML files in your \$HOME/.mavproxy/ directory (note the '.' in front of mavproxy).
- In the MAVExplorger package there is a mavgraphs.xml file included (it is the one linked above)

For your own graphs you can use a file called \$HOME/.mavproxy/mygraphs.xml and put this into it to start with:

```
<graphs>
<graph name='Test/Test Graph'>
<description>My Test Roll</description>
<expression>degrees(ATTITUDE.roll)</expression>
<expression>ATT.Roll</expression>
</graph>
</graphs>
```

A few key features of the XML file are:

- you can have as many graphs as you like in a single XML file
- Each graph has a name which should be unique
- The '/' separators in the name determine where the graph appears in the menu structure
- Each graph should have a text description
- Each graph can have multiple expression. The first expression that is applicable to the current log is used in producing the graph.

because there are multiple expressions for a graph you can create one graph definition that works for both telemetry logs and dataflash logs, and works for copter, plane and rover. It also allows us to add new expressions to cope with changing field names as ArduPilot evolves.

### Reloading the graphs

When editing XML files to add graphs you don't need to exit and restart MAVExplorger to try out your new graphs. Just run the command "reload" or use the "Reload Graphs" menu item and your new graphs will be loaded into the menus.

### Contributing Graphs

One of the main reasons for the XML files in MAVExplorger is to allow community members to contribute new graphs that are useful in log analysis. If you create a set of useful graphs please send them by email to [andrew-mavexplorer@tridgell.net](mailto:andrew-mavexplorer@tridgell.net) or open a pull request against the [MAVProxy git repository](#).

For a pull request with graphs, please add the graphs to the [MAVProxy/tools/graphs](#) directory

### Finding the Particular Commit which Introduced a Bug

This article explains how to perform a firmware binary [bisection search](#). This is an efficient technique to find the particular build/commit that introduced a reproducible bug.

**Tip**

Knowing the build in which a defect was first introduced can help identify possible causes of the problem, and inform analysis of logs and other debug techniques.

Firmware builds are available from [firmware.ardupilot.org](http://firmware.ardupilot.org) for each vehicle-type: [Copter](#), [Plane](#), [Rover](#), [AntennaTracker](#). To perform a bisection search:

1. First test the firmware build half-way between the known working and failing builds. The result gives you a new “known working” or “known failing” build and halves the number of builds that must be tested.
2. Repeat this test process, each time halving the size of the test range, until the problem build has been identified.

Once you have identified the problem issue, note the build number and include this in your bug report.

**Note**

- The autotest system generates a new firmware build for every commit to the source tree (this is a slight simplification, as commits may be batched if they arrive while a previous build is being tested). This means that if an error is reproducible it is possible to identify the specific commit(s) in which it first occurs.
- Given the number of ArduPilot firmware builds it should be possible to locate the problem build/commit within around 10 tests (at time of writing).

**Debugging with GDB**

This page describes how to setup GDB on Linux to debug issues with a PX4 or Pixhawk. The specific commands were tested on Ubuntu 13.10. GDB can also be set-up on Windows but [there is an issue passing the Ctrl-C command to GDB](#) which makes it difficult to use effectively.

**Introduction**

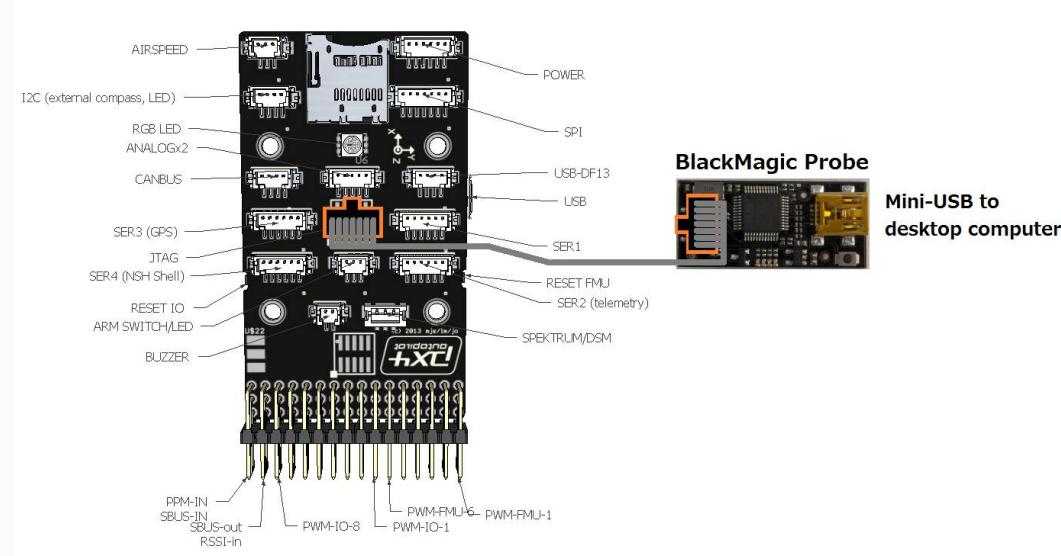
GDB (the GNU Debugger) “allows you to see what is going on ‘inside’ another program while it executes or what another program was doing at the moment it crashed.” which can be useful when investigating very low-level failures with the Pixhawk (it cannot be used with the APM1/APM2)

This guide assumes that you have already successfully built the firmware on your machine following the instructions for [Windows](#), [Mac](#) or [Linux](#).

A [BlackMagic probe](#) is also required. They can be purchased in the US from [Transition Robotics](#), [Tag-Connect](#) or [1 Bit Squared](#) or in NewZealand from [Greenstage](#).

Alternative instructions for the set-up can be found on the PX4 wiki [here](#).

**Connecting the probe to the Pixhawk**



The BlackMagic probe should be connected to the Pixhawk's JTAG connector using the grey 10wire cable that came with the probe. Note that most Pixhawk come with no headers soldered onto the JTAG connector because it interferes with the case. Please contact [Craig Elder](#) to order a Pixhawk with the JTAG pins soldered.

## Installing GDB

If using Ubuntu, GDB is likely already installed on your machine and it will likely work although we recommend using version 7.7 or higher because it includes python extensions. On an Ubuntu machine you can find where your GDB is installed by typing:

```
which gdb
```

This will likely return `/usr/local/bin/arm-none-eabi-gdb`

Next check it's version by calling it with the `--version` argument

```
/usr/local/bin/arm-none-eabi-gdb --version
```

If you wish to upgrade to 7.7 (or higher):

- download a later version from the [GDB download page](#).
- when the download completes extract the compressed file to somewhere on your machine (i.e. `/home/<username>/Documents`)
- open a terminal and change to the directory that was extracted (i.e. `/home/<username>/Documents/gdb-7.7.1`)
- run the configuration script by typing `./configure --target arm-none-eabi`
- make gdb for your system by typing `make`
- install the newly built gdb into `/usr/local/bin` by typing `make install`

## Starting GDB and running some commands

GDB requires both the binary file that's been uploaded to the board (i.e. Copter-v2.px4) which can normally be found in Copter, Plane or APMRover2 directory and the firmware.elf file that can be found in PX4Firmware/Build/px4fmu-v2\_APM.build/firmware.elf.

change to your PX4Firmware directory and type the following:

```
/usr/local/bin/arm-none-eabi-gdb Build/px4fmu-v2_APM.build/firmware.elf
```

```

rmackay9@rmackay9-Inspiron-N311z: ~/GitHub/PX4Firmware
rmackay9@rmackay9-Inspiron-N311z:~/GitHub/PX4Firmware$ /usr/local/bin/arm-none-eabi-gdb Build/px4fmu-v2_APM.build/firmware.elf
GNU gdb (GDB) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from Build/px4fmu-v2_APM.build/firmware.elf...done.
Target voltage: 3.3V
Available Targets:
No. Att Driver
 1      ARM Cortex-M
0x0809ef2a in setbasepri (basepri=16)
    at /home/rmackay9/GitHub/PX4NuttX/nuttx/include/arch/armv7-m/irq.h:227
227  __asm__ __volatile__
catching vectors: reset
Loading NuttX GDB macros. Use 'help nuttx' for more information.
Loading ARMv7M GDB macros. Use 'help armv7m' for more information.
Loading PX4 GDB macros. Use 'help px4' for more information.
Breakpoint 1 at 0x809ef42: file armv7-m/up_assert.c, line 315.
(gdb) █

```

Some useful commands:

`[r]` – restarts the process

`[b function-name]` – i.e. b setup – sets a breakpoint at the start of the “setup” function. Note a class name can be prepended such as `b AC_AttitudeControl::init`

`[Ctrl-C]` – stops the code from executing so you can set breakpoints, etc

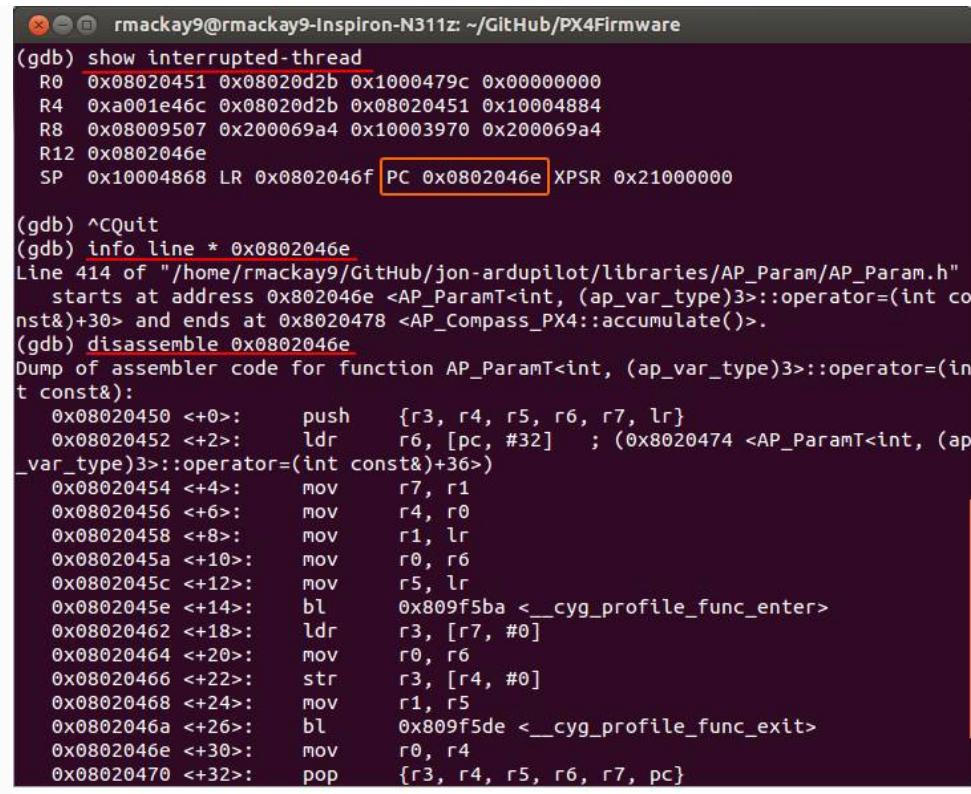
`[continue]` – continues the code from wherever it was stopped

`show interrupted-thread` – shows address where execution has stopped (see below)

`info line * <address>` – shows c++ line for a given address (i.e. from show interrupted-thread)

`disassemble <address>` – converts given address into assembler code

`[exit]` – exits from the GDB



```
r mackay9@r mackay9-Inspiron-N311z: ~/GitHub/PX4Firmware
(gdb) show interrupted-thread
R0 0x08020451 0x08020d2b 0x1000479c 0x00000000
R4 0xa001e46c 0x08020d2b 0x08020451 0x10004884
R8 0x08009507 0x200069a4 0x10003970 0x200069a4
R12 0x0802046e
SP 0x10004868 LR 0x0802046f PC 0x0802046e XPSR 0x21000000

(gdb) ^CQuit
(gdb) info line * 0x0802046e
Line 414 of "/home/r mackay9/GitHub/jon-ardupilot/libraries/AP_Param/AP_Param.h"
  starts at address 0x0802046e <AP_ParamT<int, (ap_var_type)3>::operator=(int const&)+30> and ends at 0x08020478 <AP_Compass_PX4::accumulate()>.
(gdb) disassemble 0x0802046e
Dump of assembler code for function AP_ParamT<int, (ap_var_type)3>::operator=(int const&):
0x08020450 <+0>:    push    {r3, r4, r5, r6, r7, lr}
0x08020452 <+2>:    ldr     r6, [pc, #32] ; (0x08020474 <AP_ParamT<int, (ap_
var_type)3>::operator=(int const&)+36>)
0x08020454 <+4>:    mov     r7, r1
0x08020456 <+6>:    mov     r4, r0
0x08020458 <+8>:    mov     r1, lr
0x0802045a <+10>:   mov     r0, r6
0x0802045c <+12>:   mov     r5, lr
0x0802045e <+14>:   bl     0x809f5ba <__cyg_profile_func_enter>
0x08020462 <+18>:   ldr     r3, [r7, #0]
0x08020464 <+20>:   mov     r0, r6
0x08020466 <+22>:   str     r3, [r4, #0]
0x08020468 <+24>:   mov     r1, r5
0x0802046a <+26>:   bl     0x809f5de <__cyg_profile_func_exit>
0x0802046e <+30>:   mov     r0, r4
0x08020470 <+32>:   pop    {r3, r4, r5, r6, r7, pc}
```

## Debugging using JTAG

### Overview

The JTAG interface on APM provides additional debugging features that can be useful when working on certain kinds of problems. This page describes how to configure your APM setup so that you can use JTAG to debug it.

Note that enabling JTAG disables the ADC4, ADC5, ADC6 and ADC7 pins. These aren't normally used by APM, but if your application requires more than the low four ADC inputs, you won't be able to use JTAG.

### Hardware Required

Obviously you will need an APM board. The JTAG signals are available on the oilpan, so we will assume you are using one as well. For other boards (e.g. Arduino Mega) see your board's documentation for the location of the relevant pins.

To program the JTAG and OCD fuses in the ATMega1280 an ISP programmer of some sort is required. This article describes using the cheap and effective !BusPirate, but another programmer (e.g. STK500, GoodFET, etc.) could also be used.

Last but not least, you will need an AVR JTAG unit. The Atmel JTAGICE mkii works well, but is expensive. This article was developed using the much cheaper JTAGICE mkii-CN from

<http://www.mcuzone.com>

obtained via eBay for about 1/3 the price of the Atmel unit. The JTAGICE can be used as a programmer as well.

Note that the mkii-CN unit seems very sensitive to USB setup. If it misbehaves when connected to a hub,

try connecting it directly to a port on your system.

## Software Required

The AVRDUDE and AVaRICE tools are required, as well as avr-gdb.

On Windows, all of the required tools should be installed by WinAVR. (TBD: verify whether AVaRICE comes with WinAVR)

On Mac OS the use the MacPorts tool to install the tools. Fetch and install the MacPorts package, then use the following commands:

```
sudo port selfupdate
sudo port install avrdude
sudo port install avarice
sudo port install avr-gdb
```

You can omit avr-gdb if you already have the AVR CrossPack installed.

Most Linux systems have packages available for avrdude, avarice and avr-gdb. See your system management documentation for details on installing these, or ask a friend.

## Initial Setup

Before debugging, the JTAG Enable and On Chip Debug Enable fuses must be programmed in the ATMega1280. By default both are disabled, and despite offering a tempting option in the boards.txt file, Arduino does not seem to provide a way to change them.

Separate the APM and oilpan boards, and connect your ISP programmer to the AT1280 SPI header on the APM board. In the illustration here, the BusPirate breakout cable is being used. Note that the wire colours shown are for the cable supplied by !SparkFun; other vendors may use different colour-coding. Use the following table as a guide:

!BusPirate Pin	ISP Pin	Wire Colour
1 (GND)	6	black
3 (+5)	2	grey
7 (CLK)	3	yellow
8 (MOSI)	4	orange
9 (CS)	5	red
10 (MISO)	1	brown

Programmer type should be BusPirate.

Here is a table of connections if you are using the JTAGICE mkII-CN debugger as an ISP programmer:

JTAGICE Pin	ISP Pin
TDO	1
Vref	2
TCK	3
TDI	4
SRST	5

GND	6
-----	---

Programmer type should be jtagmkii. NOTE: this configuration has not been verified to work.

Use this AVRDUDE command to program the fuses on the ATMega1280 to enable OCD and JTAG:

```
avrduude -P
-c
-p m1280 -U hfuse:w:0x1a:m -v
```

Replace

with the port to which your programmer is connected, and

with the programmer type.

This should only take a few seconds, and you should see towards the end the following lines output:

```
avrduude: safemode: lfuse reads as FF
avrduude: safemode: hfuse reads as 1A
avrduude: safemode: efuse reads as F5
avrduude: safemode: Fuses OK
```

These settings are persistent; unless you reflash the bootloader on your APM JTAG will remain permanently enabled.

To revert to the default fuse settings, use:

```
avrduude -P
-c
-p m1280 -U hfuse:w:0xda:m
```

## JTAG Hardware Setup

On the oilpan, add a 2x3 header to the Expansion Ports pads, and a 1x7 header to the analog pads (GND-AN5).

Connect your JTAG debugger as follows. Note that in the example here a SparkFun !BusPirate breakout cable is being connected to a JTAGICE mkii-CN. Your wire colours may vary; use the following table as a guide:

JTAG Pin	Oilpan Pin	Wire Colour
TCK	ADC4	brown
TMS	ADC5	green
TDO	ADC6	orange
TDI	ADC7	white
GND	GND	black, red
Vref	+5	yellow
SRST	Reset	blue

## Software Setup

Once the hardware setup is done, there are a variety of ways to debug using JTAG. The following steps cover just the basics.

The bridge between the JTAG unit and your debugger is AVaRICE. Some setups will automatically launch the tool when the debugger starts, others require that you launch it manually. It is critical for successful APM debugging that the `--capture` argument is passed to AVaRICE, rather than attempting to download a new program.

To start AVaRICE manually, assuming a JTAGICEmkII (or clone) connected via USB, use:

```
avarice --mkII --capture --jtag usb :4242
```

You should see output like:

```
AVaRICE version 2.10, Dec 22 2010 21:38:18
```

Defaulting JTAG bitrate to 250 kHz.

JTAG config starting. Found a device: JTAGICEmkII Serial number: 00:a0:00:40:26:63 Reported JTAG device ID: 0x9703 Configured for device ID: 0x9703 atmega1280 JTAG config complete. Waiting for connection on port 4242.

Note that AVaRICE will exit every time the debugger disconnects, so be prepared to restart it regularly if your debugger doesn't manage it already. Once the "Waiting for connection" message is displayed, you are ready to connect with your debugger.

## Debugging Code Built by Arduino

Arduino does its best to hide the dirty parts of the build process from the user. This can make it slightly difficult to find the files that you need for debugging. Hold down the SHIFT key while clicking the Verify button in the Arduino IDE, and you will see the commands executed by Arduino as it builds your sketch. Assuming that the build is successful, the last command listed (before the "Binary sketch..." line) will look something like this example taken from a Mac OS system:

```
/Volumes/Data/Applications/Arduino.app/Contents/Resources/Java/hardware/tools/avr/bin/avr-objcopy -O ihex
-R .eeprom
/var/folders/Bu/Burcn-0aFa4N+++0Me1I2U++1dw/-Tmp-/build159308262941509457.tmp/GPS_AUTO_test.cpp.elf
/var/folders/Bu/Burcn-0aFa4N+++0Me1I2U++1dw/-Tmp-/build159308262941509457.tmp/GPS_AUTO_test.cpp.hex
```

The file that your debugger will want to load is the .elf file, in this case

```
/var/folders/Bu/Burcn-0aFa4N+++0Me1I2U++1dw/-Tmp-
/build159308262941509457.tmp/GPS_AUTO_test.cpp.elf
```

The name of the file will depend on your sketch name, and the location will vary depending on your system, but it will remain the same for a given sketch until you quit and restart Arduino. Note that the directory containing the file is deleted and re-created every time you compile the sketch.

To debug this file with avr-gdb, use Arduino to compile and upload the sketch to APM. Start AVaRICE as described above, then start GDB:

```
avr-gdb /var/folders/Bu/Burcn-0aFa4N+++0Me1I2U++1dw/-Tmp-/build159308262941509457.tmp/GPS_AUTO_test.cpp.elf
```

You should see GDB start up:

```
GNU gdb 6.8
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later

This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i386-apple-darwin9.8.0 --target=avr"...
(gdb)
```

At the (gdb) prompt, tell GDB to connect to AVaRICE:

```
(gdb) target remote localhost:4242
```

This will stop the program running on the APM and tell you where it stopped:

```
Remote debugging using localhost:4242
0x00002992 in AP_GPS_NMEA::read (this=0x8007aa)
at /Volumes/Data/Users/msmith/work/Mike/ArduPilot/Sketchbook/libraries/AP_GPS/AP_GPS_NMEA.cpp:72
72      numc = _port->
available();
(gdb)
```

Use the ‘continue’ command to start the program running again, and hit control-C to stop it once more.

At this point, GDB can be used normally. Other debuggers that are compatible with or build on GDB such as DDD or Insight can be used instead, as can plugins for various IDEs like Eclipse.

## Interfacing with Pixhawk using the NSH

This article explains how to communicate with a Pixhawk using the [NuttX Shell \(NSH\)](#) using either a serial or remote connection.

### Overview

The Pixhawk runs the NuttX real-time operating system which includes the NuttX Shell terminal “NSH”. This allows running some Unix style commands including “top” and “ls”.

NSH is very useful for diagnosing low level issues. Some of the things you can do with it include:

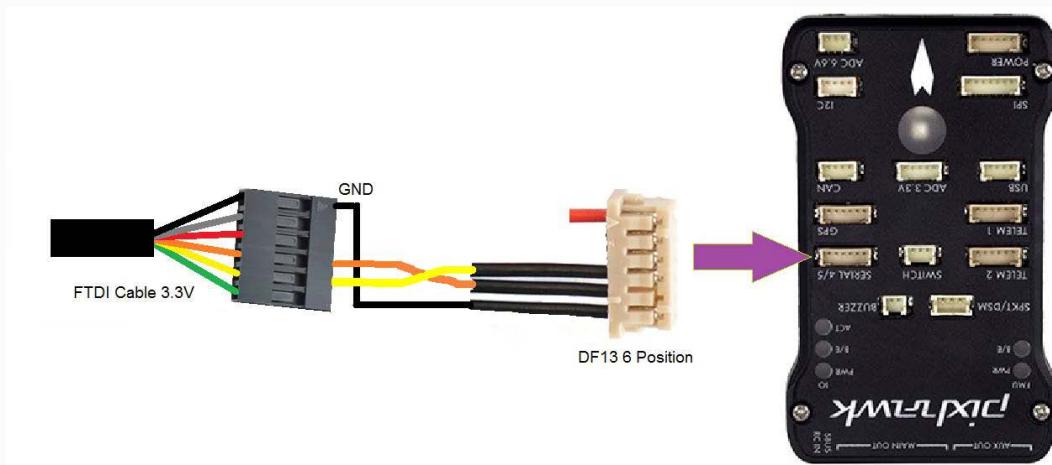
- Display performance counters with the `perf` command
- Display px4io status information
- Diagnose microSD errors
- Diagnose sensor failures
- Assist in debugging new drivers

This topic explains how to connect and send NSH commands from MAVProxy via remote and serial connections.

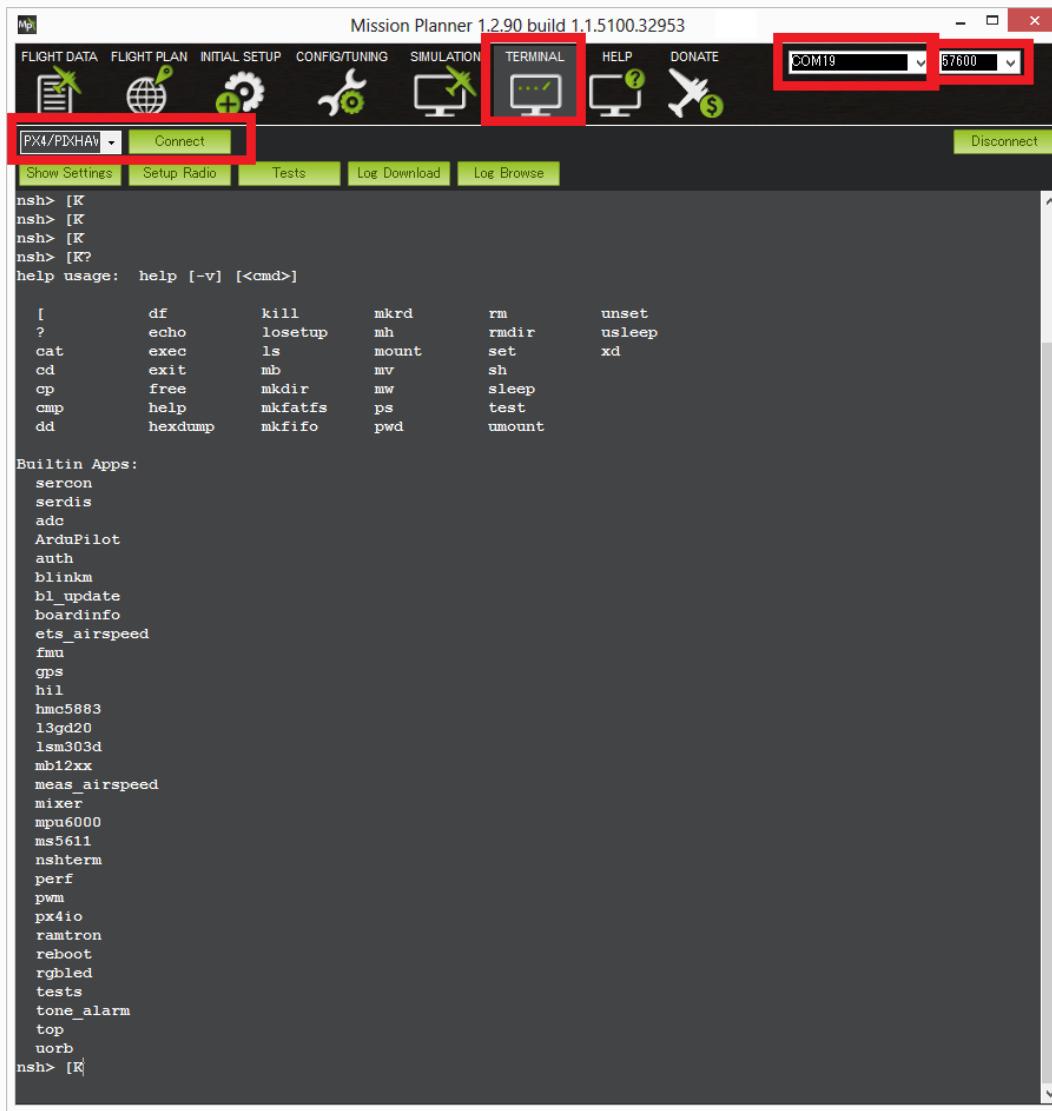
### Running NSH using Debug Cable and Serial5

To use the NSH while Copter or Plane is running you can connect using Serial 4/5. To do this you will need an [FTDI 3.3V cable](#) and then modify a [DF13 6 Position cable](#) so that it can be connected to the FTDI

cable.



You should then be able to plug the FTDI cable into your computer and connect with any Serial program including the Mission Planner's Terminal screen. Ensure to select the FTDI cable's COM port and set the Baud rate to 57600. You will need to press return for the "nsh>" prompt to appear.



## Remote NSH over MAVLink

ArduPilot also includes support to run nsh commands remotely via MAVLink over a USB, telemetry or

WiFi link (this is an extension of the SERIAL\_CONTROL protocol used for controlling a GPS or radio UART over MAVLink).

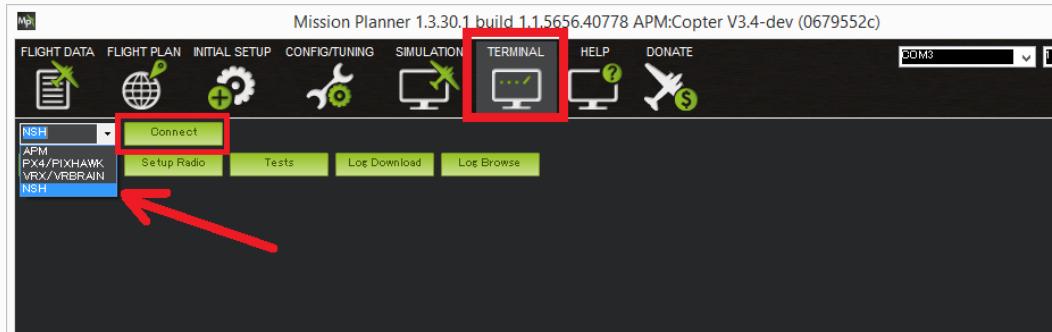
#### Note

At time of writing Rover doesn't support NSH over MAVLink (because it doesn't support arming).

#### Tip

This can be used for nsh debugging when you don't have a serial5 cable setup.

#### Instructions to use with Mission Planner:



- Connect Pixhawk to PC using a USB cable
- Go to the Flight Data screen, select the correct COM port and baud (probably 115200) and press the connect button
- Go to the Terminal screen, set the left-most drop down to "NSH" and push Connect.

#### Setup instructions to access using MAVProxy:

1. Load latest master onto a Pixhawk

2. Grab the latest MAVProxy

- If on Linux use:

```
sudo pip install mavproxy
```

- If on Windows [grab it here](#).

#### Note

##### If you are on Windows you will need to delete any old

version of MAVProxy in **c:\Program Files (x86)\MAVProxy** first, as the installer does not properly cleanup old versions.

3. Start MAVProxy as usual

4. Load the nsh module in MAVProxy with

```
module load nsh
```

5. start the nsh shell with

```
nsh start
```

6. Now run nsh commands as usual. Note that MAVLink is still running, so the map, *MAPProxy* console, graphs etc all keep updating while in the nsh shell.

#### Tip

A blank response to an NSH command may indicate low RAM memory on the autopilot board. Free up memory if possible. For example, on master you can save about 20Kb by disabling terrain following (set `TERRAIN_ENABLE=0`).

7. To drop back to the normal MAVProxy prompt type a single "." on a line by itself

You can only start a shell when the system is disarmed. Once the shell is started you can arm if you like.

#### Warning

In theory you could fly while doing nsh commands, but we don't recommend it.

This also works over a 3DR Radio link, although it is of course slower in output than when on a USB connection.

## Using Linux Trace Toolkit ng (LTTng) to Create Realtime ArduPilot Traces

This article explains how to use [LTTng](#) for creating runtime traces for ArduPilot running on Linux boards (only).

#### Note

This support in the master branch (Jan 2016) but is not yet in the released vehicle branches (e.g. not in Copter 3.2, Plane 3.4 or Rover 2.5)

### Overview

Tracing is a technique used to record the real-time behaviour of a software in memory and analyze the execution of the code off-line. It can be very useful to understand the interaction between threads, processes, the duration of certain events and the sequencing of some operations.

*LTTng* is a Linux tool that can be used to trace either a userspace application, or the Linux kernel itself. It produces trace files that can be analyzed later, mostly off-board. Lttng's implementation in APM uses userspace tracing but you can at the same time monitor what's going on inside the kernel, provided the kernel has tracepoints enabled (in its config).

*LTTng* has been integrated using the [Perf\\_Ltng](#) class, which handles 3 operations: `Begin`, `End`, `Count`. You add tracing to your code by calling the performance utility methods in [Perf.cpp](#).

#### Note

The only modules that currently include calls to the Perf API are EKF and EKF2 but it is possible to use it to monitor other modules.

### Monitoring APM with LTTng

In order to monitor APM with *LTTng*, the first thing to do is to enable support for *LTTng* in ArduPilot builds. Using the regular make build system, you do this by modifying [mk/board\\_native.mk](#) line 30:

```
HAVE_LTNG=1
```

Then clean and rebuild ArduPilot.

#### Note

You can't build statically with *LTTng* enabled because it calls dlopen. On the bebop, I had to build the whole system with armhf toolchain, which is not the one used by default

### Putting trace events inside the code

In order to trace ArduPilot, you call the following performance utility methods:

```
hal.util->perf_begin(my_perf);
```

```
hal.util->perf_end(my_perf);
```

Before calling these functions, `my_perf` needs to be allocated

```
AP_HAL::Util::perf_counter_t my_perf = hal.util->perf_alloc(AP_HAL::Util::PC_ELAPSED, "my_perf");
```

An example of adding multiple perf events can be seen in [this patch](#).

## Installing LTTng on your board

In order to use *LTTng*, you will have to install *LTTng* tools on your board.

The *LTTng* documentation explains how to install it on your Linux distribution [here](#). If there is no official package for your Linux distribution, you can also build it from the sources as explained [here](#)

## Trace capture

Enable trace events in ArduPilot

Once you have the modified version of ArduPilot launched on your board and *LTTng* tools installed, you can check that ArduPilot *LTTng* events are available on your board's console terminal using the following command:

```
lttng list --userspace
```

This should output a list of events including:

```
ardupilot:count (loglevel: TRACE_DEBUG_LINE (13)) (type: tracepoint)
ardupilot:end (loglevel: TRACE_DEBUG_LINE (13)) (type: tracepoint)
ardupilot:begin (loglevel: TRACE_DEBUG_LINE (13)) (type: tracepoint)
```

Tip

Refer to the [LTTng documentation](#) for information on how to list kernel events.

Capturing a trace session

A trace session can be started, stopped, re-started and destroyed.

Destroying a trace session doesn't delete the traces from permanent storage but means that you have to restart a new session if you want to continue capturing.

1. Create session:

```
lttng create -o my_trace_directory
```

2. Enable ArduPilot events:

```
lttng enable-event --userspace ardupilot:count
lttng enable-event --userspace ardupilot:end
lttng enable-event --userspace ardupilot:begin
```

To enable kernel events, see [these instructions](#).

### 3. Start capturing:

```
lttng start
```

### 4. To stop capturing:

```
lttng stop
```

### 5. If you have *babeltrace* installed on board, you can view the result in text:

```
lttng view
```

### 6. Restart or destroy your session:

```
lttng destroy
```

## Analyzing the trace

In order to analyze your trace, first copy the trace directory you indicated at session creation to your computer.

### Babeltrace

You can analyze the trace using *babeltrace*. This is a command line tool that is able to translate the trace into text format:

```
babeltrace my_trace_directory
```

The result is a series of events with the according timestamps and the number of the CPU they have been running on.

To go further with that, you can use [babeltrace's python bindings](#) to write a python program analyzing the trace you have captured.

### Trace compass

[Trace compass](#) is an Eclipse plugin that can read *LTTng* traces and display them in a UI that is adapted to some analyzes.

### Lttng2lxt

Trace compass is very powerful but up to now, I haven't been able to analyze userspace traces in a convenient way.

*Lttng2lxt* is an open source tool written by Ivan Djelic from Parrot. It is a very simple command line tool that generate waveforms readable by [gtkwave](#). I have added support for ArduPilot events to *Lttng2lxt*.

In order to use it, get the sources from my github repository:

```
git clone https://github.com/jberaud/lttng2lxt
```

You will have to install *libbabeltrace-ctf-dev* and *libbabeltrace*. On Debian or Ubuntu do this with the following command:

```
sudo apt-get install libbabeltrace-ctf-dev libbabeltrace
```

Compile it:

```
sudo make install
```

Use it to generate a *gtkwave* waveform file:

```
lttng2lxt my_trace_directory
```

This will produce a file called **my\_trace\_directory.lxt**.

## Analyzing your trace using gtkwave

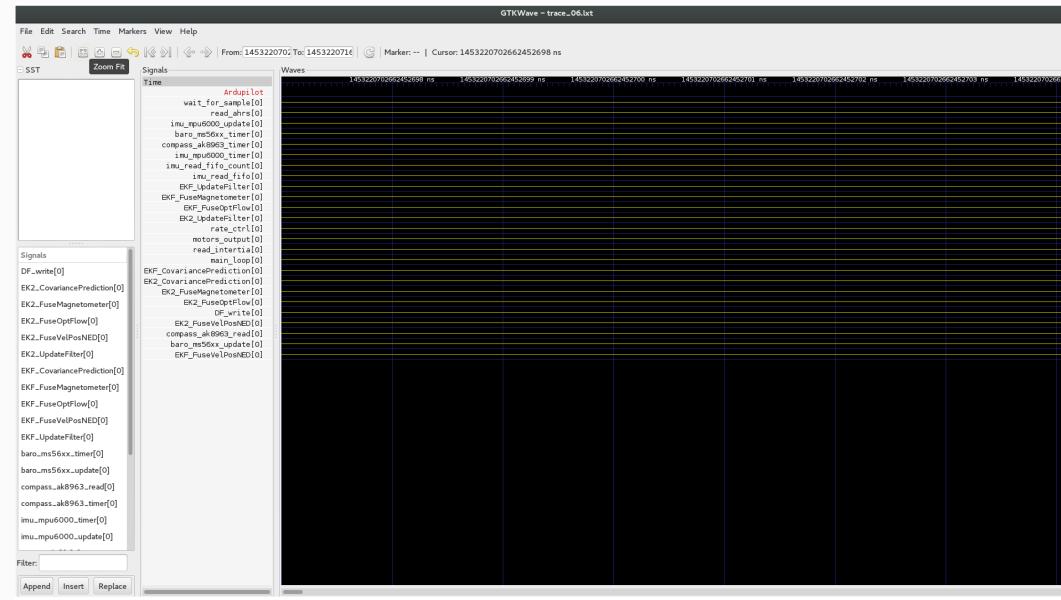
First install *gtkwave*. On Debian or Ubuntu you can do this with the command:

```
sudo apt-get install gtkwave
```

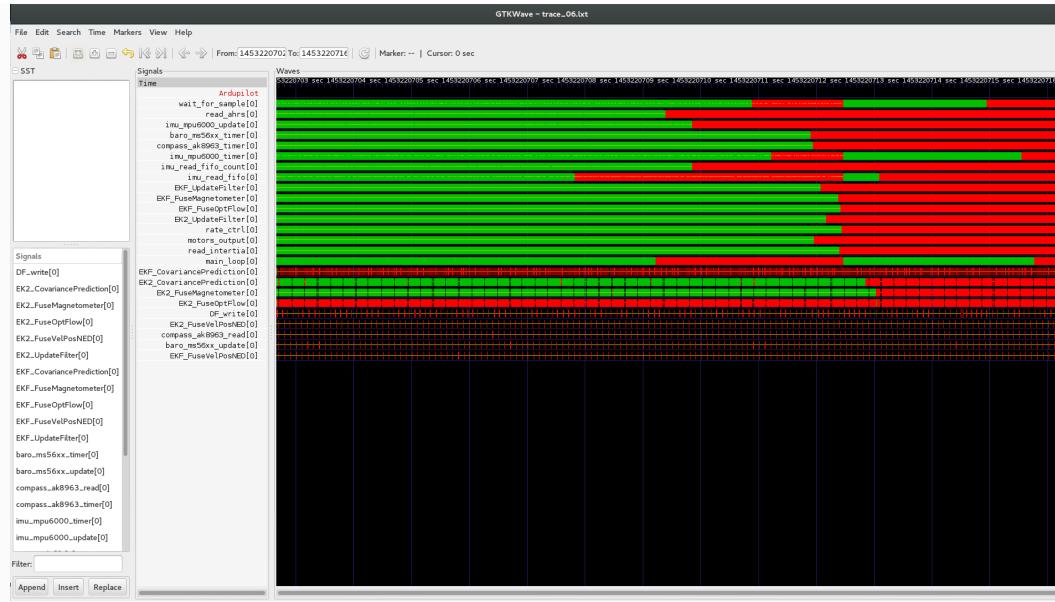
Then open your trace using *gtkwave*:

```
gtkwave -A my_trace_directory.lxt
```

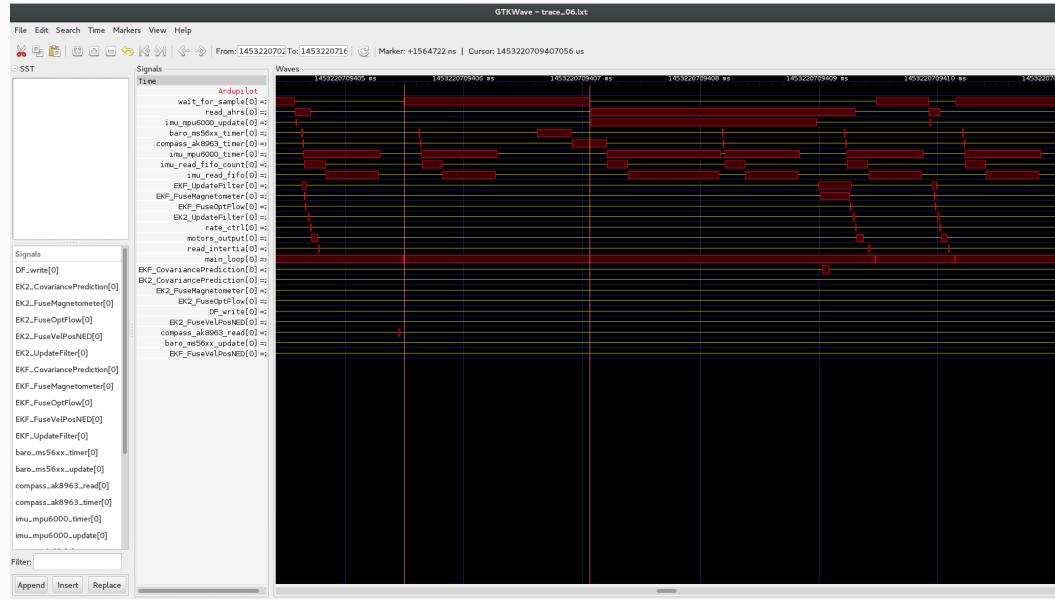
### Use the **Zoom fit** button



The result should look like this:



You can then zoom in or out and see the exact execution sequence, with the associated timings



You can therefore measure the duration of an event with the cursors (use the mouse middle button to set the first cursor)

## Loading a bootloader with DFU

This page describes how to load a new bootloader onto a STM32 based board (such as a Pixhawk1) via DFU. This is useful if you are either trying to bring up a new board or if you have a corrupted bootloader on an existing board.

### What is DFU

DFU is the “Direct Firmware Update” mode for some microcontrollers, most notably the STM32F4 series. It allows you to load a firmware (including a bootloader) over USB using widely available DFU utilities.

### Accessing DFU Mode

You access it by pulling the “boot0” pin high on the processor when it is powered on. On a Pixhawk1 this is done by pulling the “FMU-BOOT” pad on the top surface of the Pixhawk1 high. The FMU-BOOT pad is located between the buzzer and DSM/Spkt connectors on the Pixhawk1. You should pull it up to 3.3V, but in a pinch it does work to pull it up to 5V if you don’t have 3.3V handy.

When the pin is pulled up apply power to the board (eg. plug in the USB connector) and the board should boot into DFU mode.

## **dfu-util tool**

It is recommended that you install the dfu-util tool. On Linux machines with apt you can do that with:

```
sudo apt-get install dfu-util
```

On other systems please see <http://dfu-util.sourceforge.net/>.

## **Listing DFU Devices**

Run the following:

```
dfu-util --list
```

You should get a result like this:

```
dfu-util --list
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

Found DFU: [0483:df11] ver=2200, devnum=49, cfg=1, intf=0, alt=3, name="@Device Feature/0xFFFF0000/01*004
e", serial="315A35663432"
Found DFU: [0483:df11] ver=2200, devnum=49, cfg=1, intf=0, alt=2, name="@OTP Memory /0x1FFF7800/01*512
e,01*016 e/0x1FFE7800/01*512 e,01*016 e", serial="315A35663432"
Found DFU: [0483:df11] ver=2200, devnum=49, cfg=1, intf=0, alt=1, name="@Option Bytes /0x1FFFC000/01*016
e/0x1FFEC000/01*016 e", serial="315A35663432"
Found DFU: [0483:df11] ver=2200, devnum=49, cfg=1, intf=0, alt=0, name="@Internal Flash
/0x08000000/04*016Kg,01*064Kg,07*128Kg,04*016Kg,01*064Kg,07*128Kg", serial="315A35663432"
```

If you don’t get that then do some googling on how to debug USB connection issues with DFU.

## **Loading a bootloader**

The current bootloaders suitable for ArduPilot on STM32 are here:

<https://github.com/ArduPilot/ardupilot/tree/master/mk/PX4/bootloader>

download the px4fmuv2\_bl.bin and run this:

```
dfu-util -a 0 -dfuse-address 0x08000000 -D px4fmuv2_bl.bin
```

it should say “Downloading” and show a progress bar. On completion the board is ready to test the bootloader.

After you have the bootloader loaded power cycle with the boot0 pin pulled down (note that it is already pulled down by a resistor on a Pixhawk1, so just power cycle).

Then check your USB bus and you should see a device “PX4 BL FMU v2.x” with vendorID 0x26ac and productID 0x0011. You can now use the normal firmware load tools from ArduPilot to load a flight firmware.

## Contributing

This section contains topics related to coding standards, release procedures and contributing to the ArduPilot project.

### Guidelines for Contributors to the APM Codebase

**How to become a contributor:** Please see [this post](#) for full details. Short version: we are always looking for new team members, but to avoid disrupting existing work or destabilizing code, we have processes that bring people in gradually. In general the process is:

- Find a specific bug you'd like to fix or a specific feature you'd like to add (check out the [issues list](#) if to get some ideas).
- Fix the bug in your own clone and test that it's working
- Submit the change to the main code base [via a pull request](#).

If this fits in well with the project's direction, you will be added to the list of developers and added to the weekly dev call.

### Submitting Patches Back to Master

Once you have a bug fix or new feature you would like to have included in the APM projects, you should submit a [Pull Request](#).

The main developers will see your changes in the [Pulls list](#), review them and if all goes well they will be merged into master.

### Preparing commits

Commits should be made to your fork/clone of the project in a new branch (i.e. not “master”) that is up-to-date with the ardupilot/ardupilot master branch does and not include any other changes. See [Working with the ArduPilot Project Code](#) for instructions on how to correctly update your working branch.

**Commits should be small, and do just one thing.** If a change touches multiple libraries then there should be a separate commit per library, and a separate commit per vehicle directory. This is true even if it means that intermediate commits break the build.

Unix line endings (LF) are used. Git should take care of this automatically, but if you notice that you have a lot of files that show up as changed in [git status](#) but you didn't touch those files, you may need to [check to see if local git settings regarding line endings are correct](#).

**Well-written, concise comments are good** and are encouraged.

Do not submit patches with commented-out code, or code that is never reachable within `#define` s.

Before submitting code to the official repository, **clean up your local commit history**. When submitting patches the convention is to [use an interactive rebase](#) (eg. “git rebase -i HEAD~10”) to re-arrange patches and fold things together, so the patch set is “cleaned up” for submission. The idea is to present a

logical set of patches for review. It can take a bit of effort to get used to interactive rebase, but it is definitely worth learning. Refer to [online resources](#) to understand how to use this tool.

Try to follow the style conventions of the existing code. In particular, check that your [editor uses 4 spaces instead of tab](#).

## Log messages

Commit messages should be of the form:

```
Subsystem: brief description
```

```
Longer description...
```

Example:

```
APM_Control: reduce the number of parameter saves in autotune
```

```
don't save a parameter unless it has changed by 0.1%
```

## Before you submit a pull request

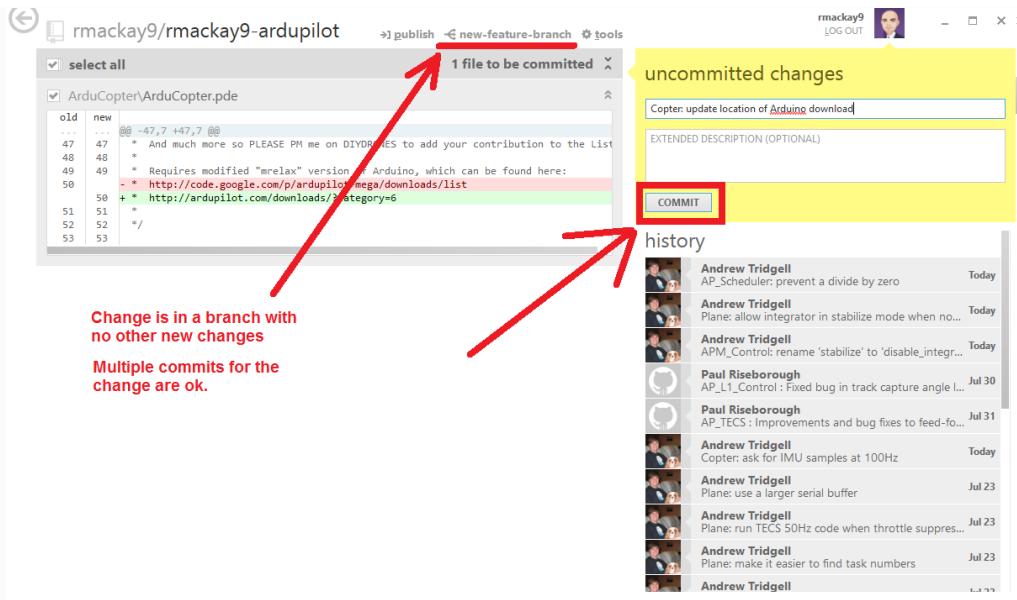
Before submitting a pull request please do the following

- Use git rebase on current master. Your changes should not include any merges, and should contain just the patches you want as the last patches in your tree.
- read your changes, doing your own review. The best way to do this is to use the “gitk” tool. Look over your own changes critically. Make sure they don’t include anything you don’t want to go into the pull request. It is best to read your changes at least several hours after you wrote the code, and preferably the next day. Look over them carefully and look for any bugs.
- if you have access to a Linux build environment then build your modified tree using Tools/scripts/build\_all.sh. That will test that all the builds for different boards and vehicle types work. If you don’t have a Linux build environment then please test the build for APM2 and PX4 and rover, copter and plane if your changes may affect those environments.
- test your changes in SITL if possible. If you can’t run SITL then test your changes in a real vehicle.

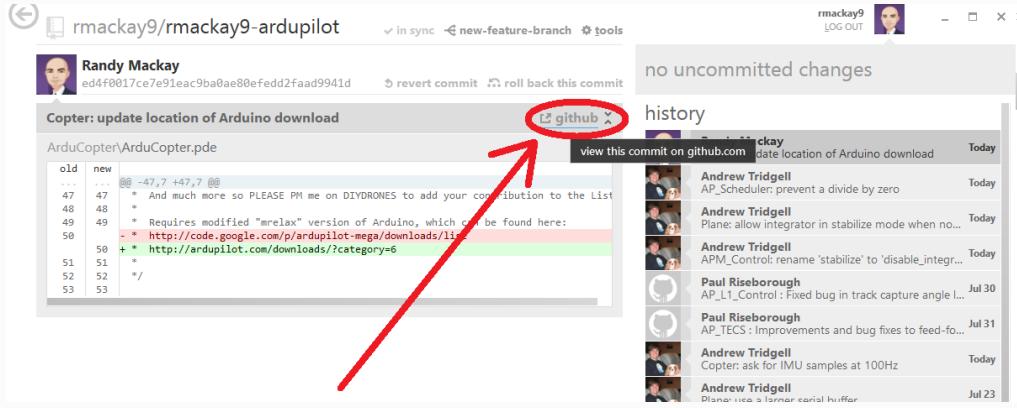
## Submitting pull requests

To submit a patch for review and possible inclusion in the official repository, follow these directions:

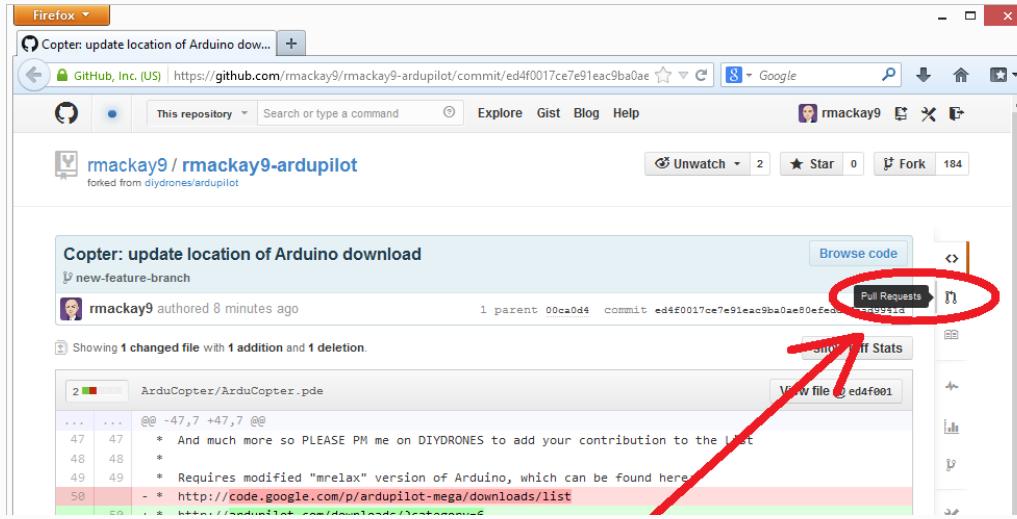
1. Clean up your local commit history with `git rebase -i` as described above.
2. Push your local branch to GitHub. If using the GitHub for Windows client, this screenshot shows what your commit/sync screen could look like:



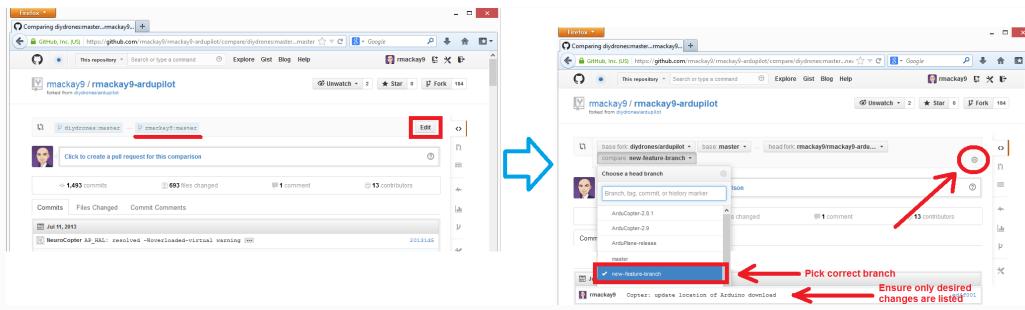
3. Open your clone's repository on the GitHub web page and [Create a pull request on GitHub](#). You'll be making a pull request from your fork/branch to the ardupilot/master repository. If using the GitHub for Windows client, one convenient way to navigate to the repository/branch is to click one of your commits and click the "github" (view this commit on github.com) button:



4. On the right side of the web page select "Pull Request", and then select the green "New pull request" button:



5. The comparison should be between ardupilot:master and the new branch you created for the feature but it has probably defaulted to your clone's master branch so click the "Edit" button and change it to the correct branch:



6. Check the list of change at the bottom of the page only includes your intended changes, then press "Click to create pull request for this comparison".
7. Let the dev team know! Post on the [drones-discuss](#) mailing list so the discussion and code review is easily publicly available.

It is very common, especially for large changes, for the main developers to ask you to modify your pull request to fit in better with the existing code base or resolve some knock-on impact that you may not have known about. Please don't take this the wrong way, we're definitely not trying to make your life difficult!

## Release Procedures

This page outlines the steps that are normally followed for a Copter release. Listed mostly as a reference so that we don't forget the steps.

### Alpha Testing

The [AutoTester](#) that runs after each commit and highlights issues that it's been setup to test.

Developers and some Alpha testers perform intermittent tests of master especially after new features have been added.

### Beta Testing Release Candidates

Release candidates are made available to beta testers through the mission planner's Beta Firmware's link. The Mission Planner Beta Firmware link pulls the version from the [Copter/beta directory of firmware.ardupilot.org](#).

Someone with ardupilot GitHub commit access (normally Randy) makes the new firmware available through the following steps:

#### For Pixhawk/PX4:

Open a Git Bash terminal in the ardupilot repository:

a) `git checkout master` (or the ArduCopter-3.1.2 branch if this is a release candidate for a patch release)

b. update ArduCopter.pde's firmware version and ReleaseNotes.txt

in ardupilot, PX4Firmware and PX4NuttX directories:

c) `git show ArduCopter-beta` and record the old git tags in case a backout is required

d. `git tag -d ArduCopter-beta`

e. `git push origin :refs/tags/ArduCopter-beta`

f. `git tag ArduCopter-beta HEAD`

g. `git push origin ArduCopter-beta`

#### For APM1/APM2:

Open a Git Bash terminal in the ardupilot repository:

a) `git checkout master` (or the ArduCopter-3.1 branch if this is a release candidate for a patch release)

b) update ArduCopter.pde's firmware version and ReleaseNotes.txt (if not already done above for Pixhawk)

in the ardupilot directory:

c) `git show ArduCopter-beta-apm1` and record the old git tags in case a backout is required

d. `git show ArduCopter-beta-apm2`

e. `git tag -d ArduCopter-beta-apm1`

f. `git tag -d ArduCopter-beta-apm2`

g. `git push origin :refs/tags/ArduCopter-beta-apm1`

h. `git push origin :refs/tags/ArduCopter-beta-apm2`

i. `git tag ArduCopter-beta-apm1 HEAD`

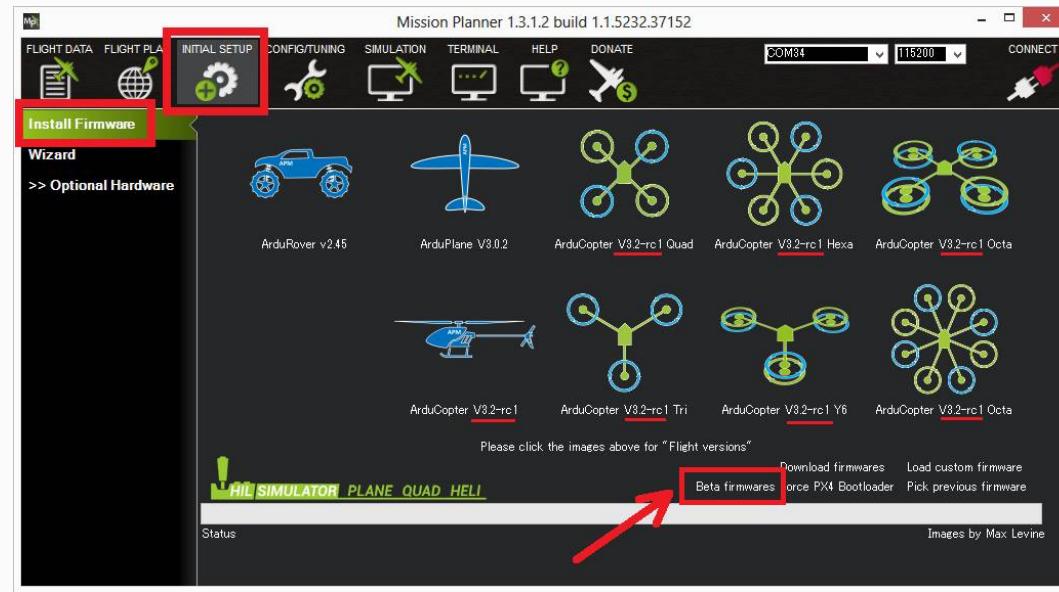
j. `git tag ArduCopter-beta-apm2 HEAD`

k. `git push origin ArduCopter-beta-apm1`

l) `git push origin ArduCopter-beta-apm2` Commit a small change ([like this one](#)) to the ardupilot directory which will cause the firmware to be rebuilt.

#### Check the versions are available

Open the Mission Planner's Initial Setup > Install Firmware page and click the "Beta firmwares" link and ensure that the version displayed below each multicopter icon has updated.



#### Communicating the new release candidate

Let Beta testers know the new version is available by emailing the [drones-discuss@googlegroups.com](mailto:drones-discuss@googlegroups.com) and [arducoptertesters@googlegroups.com](mailto:arducoptertesters@googlegroups.com). Include the contents of the ReleaseNotes.txt so people know what has changed.

Post a note into the latest “Copter Released!” thread ([like this one for AC3.1](#)) and if this is the first release candidate for a new version create a new discussion on DIYDrones ([like this one for AC3.2](#)) to concentrate the discussion.

## Issue Tracking

Issues reported by Beta Testers via the DIYDrones discussion or the arducoptertesters email list are tracked by the lead Tester (Marco) and developer (Randy) perhaps using a google spreadsheet ([like this one](#)). They are then investigated by one of the developers (usually Randy, Leonard, Jonathan or Rob) and if it's determined to be an issue it's added to the regular [Issues List](#). Sometimes beta testers directly log an issue into the issues list and this is ok but it risks the issues list becoming a support forum so it's better that it is investigated before being added to the issues list. Someone with edit access on the issues list (usually Randy) should set the “Milestone” on the issue to the current release if we plan to resolve the issue before the official release.

## Final Release

### Branching and tagging before the release

Approximately 2 to 3 weeks before the final release a new branch is created in the ardupilot repository. This protects the final release from changes that could invalidate the testing and also avoids disrupting development for future releases.

In a Git Bash terminal in the ardupilot repository:

```
git checkout -b ArduCopter-3.2
```

 (replace “ArduCopter-3.2” with the new release name)

```
git push
```

Tags are added to the PX4Firmware and PX4NuttX repositories so that we can be sure which commits were included in the release:

```
git tag ArduCopter-3.2
```

 (replace “ArduCopter-3.2” with the new release name)

```
git push origin ArduCopter-3.2
```

### Get a video ready

Ask Marco or other members of the testing team to provide videos which will be included in the final release discussion.

### Deciding to release

The “ok to release” decision is made by the lead developer (Randy) after a discussion on the weekly dev call, a review of the outstanding issues and after getting the “ok” from the lead tester (Marco).

### Release Steps

Someone with commit access (usually Randy) does the following:

#### For Pixhawk/PX4:

Open a Git Bash terminal in the ardupilot repository:

- a) `git checkout ArduCopter-3.2` (“ArduCopter-3.2” should be replaced with the branch name for the release)

b. update ArduCopter.pde's firmware version and ReleaseNotes.txt

in ardupilot, PX4Firmware and PX4NuttX directories:

c) `git show ArduCopter-stable` and record the old git tags in case a back-out is required

d. `git tag -d ArduCopter-stable`

e. `git push origin :refs/tags/ArduCopter-stable`

f. `git tag ArduCopter-stable HEAD`

g. `git push origin ArduCopter-stable`

h) `git tag ArduCopter-3.2-px4 HEAD` (where "3.2" should be replaced with the release number)

1. `git push origin ArduCopter-3.2-px4`

## For APM1/APM2:

Open a Git Bash terminal in the ardupilot repository:

a) `git checkout ArduCopter-3.2` ("ArduCopter-3.2" should be replaced with the branch name for the release)

b) `git show ArduCopter-stable-apm1` and record the old git tags in case a back-out is required

c) `git show ArduCopter-stable-apm2` and record the old git tags in case a back-out is required

d. `git tag -d ArduCopter-stable-apm1`

e. `git tag -d ArduCopter-stable-apm2`

f. `git push origin :refs/tags/ArduCopter-stable-apm1`

g. `git push origin :refs/tags/ArduCopter-stable-apm2`

h. `git tag ArduCopter-stable-apm1 HEAD`

i. `git tag ArduCopter-stable-apm2 HEAD`

j. `git push origin ArduCopter-stable-apm1`

k. `git push origin ArduCopter-stable-apm2`

l) `git tag ArduCopter-3.2-apm HEAD` (where "3.2" should be replaced with the release number)

m. `git push origin ArduCopter-3.2-apm`

## Check the new versions are available

Open the Mission Planner's Initial Setup > Install Firmware page and ensure that the version displayed below each multicopter icon has updated.

## Communicating the Release

Let testers and developers know the release has completed by emailing the [drones-discuss@googlegroups.com](mailto:drones-discuss@googlegroups.com) and [arducoptertesters@googlegroups.com](mailto:arducoptertesters@googlegroups.com). In general there should be no changes from the final release candidate. Include the full list of changes since the last official release which can be taken from the ReleaseNotes.txt.

Create a new "ArduCopter Released!" thread ([like this one for AC3.1](#)) including videos from the beta testers and stand by for any support issues that may arise.

## Didn't find what you are looking for?

If you think of something that should be added to this site, please [open an issue](#) or post a comment on the [drones-discuss](#) mailing list.

## Ardupilot Style Guide

In order to improve the readability of the ArduPilot code and help new users pick up the code quickly please use the following styles.

### Note

Some parts of the code may not conform due to historic reasons but new additions should! `astyle` is our tool of choice for formatting the code. Whilst we aren't doing it automatically at the moment in the Tools/CodeStyle directory there is an `astylerc` file that can be used to format the code automatically according to the guidelines below.

```
astyle --options=Tools/CodeStyle/astylerc <filename>
```

Remember any formatting changes to a file should be done as a separate commit.

### Case

Variables and function names are lower case with words separated by an underscore.

#### Right:

```
throttle_input
```

#### Wrong:

```
ThrottleInput
```

### Indentation

### Spaces

Indent using 4 spaces everywhere. Do not use tabs.

#### Right:

```
int foo()
{
    return 0;
}
```

#### Wrong:

```
int foo()
{
    return 0;
}
```

### Case statements

A case label should line up with its switch statement. The case statement is indented.

### Right:

```
switch (condition) {
    case foo_cond:
        foo();
        break;
    case bar_cond:
        bar();
        break;
}
```

### Wrong:

```
switch (condition) {
    case foo_cond:
        foo();
        break;
    case bar_cond:
        bar();
        break;
}
```

## Spacing

### Unary operators

Do not place spaces around unary operators.

### Right:

```
i++;
```

### Wrong:

```
i ++ ;
```

## Control statements

Place spaces between control statements and their parentheses.

### Right:

```
if (condition) {
    foo();
}
```

### Wrong:

```
if(condition) {
    foo();
}
```

```
if (condition){
    foo();
}
```

## Function calls

Do not place spaces between a function and its parentheses, or between a parenthesis and its content.

### Right:

```
foo(a, 10);
```

### Wrong:

```
foo (a, 10);
```

```
foo(a, 10 );
```

## Trailing whitespaces

Don't leave trailing whitespace on new code (a good editor can manage this for you). Fixing whitespace on existing code should be done as a separate commit (do not include with other changes).

## Line breaks

### Single statements

Each statement should get its own line.

### Right:

```
x++;
y++;
if (condition) {
    foo();
}
```

### Wrong:

```
x++; y++;
if (condition) foo();
```

## Else statement

An `else` statement should go on the same line as a preceding close brace.

### Right:

```
if (condition) {
    foo();
} else {
    bar();
}
```

**Wrong:**

```
if (condition) {
    foo();
}
else {
    bar();
}
```

**Braces****Function braces**

Functions definitions: place each brace on its own line. For methods inside a header file, braces can be inline.

**Control statements**

Control statements (`if`, `while`, `do`, `else`) should always use braces around the statements.

**Right:**

```
if (condition) {
    foo();
} else {
    bar();
}
```

**Wrong:**

```
if (condition)
    foo();
else
    bar();
```

**Other braces**

Place the open brace on the line preceding the code block; place the close brace on its own line.

**Right:**

```
class My_Class {
    ...
};

namespace AP_HAL {
    ...
}

for (int i = 0; i < 10; i++) {
    ...
}
```

**Wrong:**

```
class My_Class
{
```

```
}; ...
```

## Names

### Private members

Private members in classes should be prefixed with an underscore:

#### Right:

```
class My_Class {
private:
    int _field;
};
```

#### Wrong:

```
class My_Class {
private:
    int field;
};
```

### Class names

Class names should capitalise each word and separate them using underscores.

#### Right:

```
class AP_Compass { };
```

#### Wrong:

```
class ap_compass { };
```

### Commenting

Each file, function and method with public visibility should have a comment at the top describing what it does.

## ArduPilot Mumble Server

The ArduPilot dev team uses a [mumble server](#) for real-time voice chat. We have a “General Discussion” channel for chatting about ArduPilot development at any time. How many developers are connected depends a lot on timezones. Don’t be shy about your English language ability, we have developers from all over the world.

#### Note

The channel is for development discussions only. Please don’t use it as a user help channel. We don’t record the calls, and we would ask you not to record them yourself.

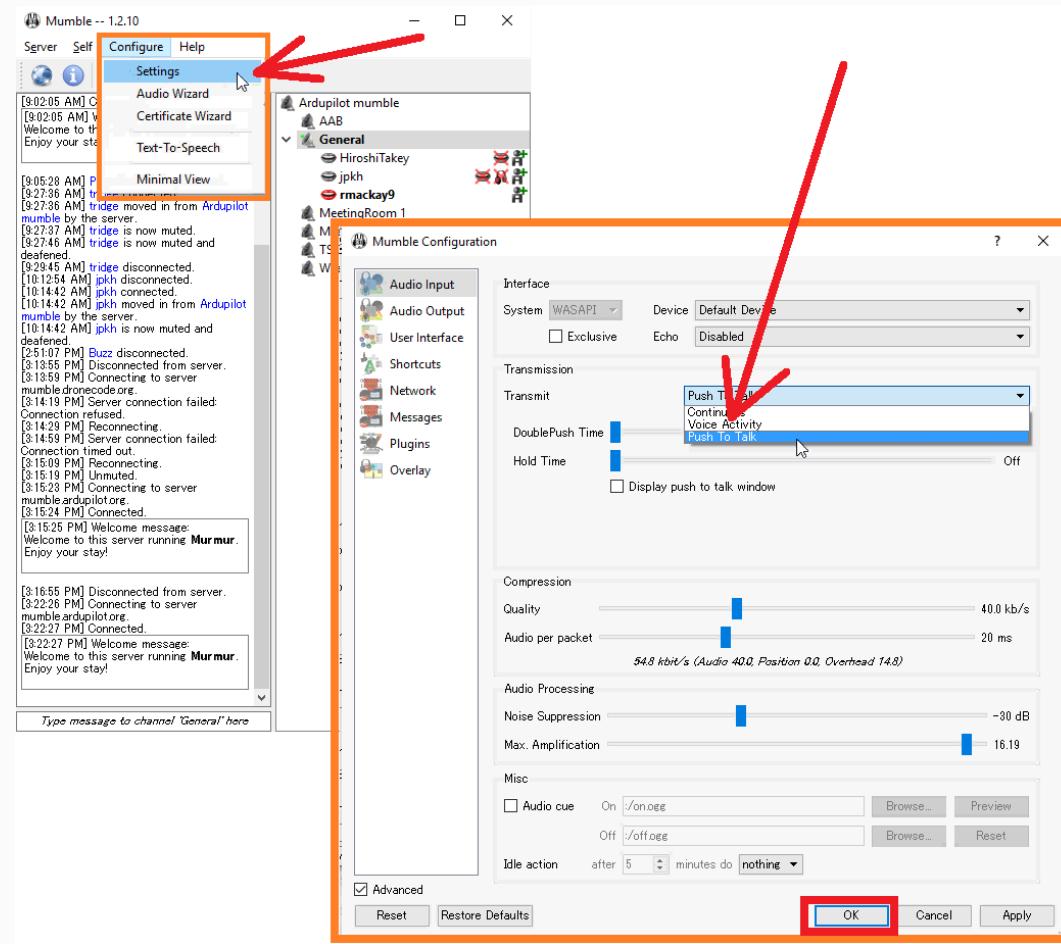
### Installing and Connecting

- Download the software from <http://mumble.sourceforge.net> (look for **Download Mumble** and pick the "Stable Release" for your operating system)
- Install and open mumble
- from the Server menu select Connect
- server address: mumble.ardupilot.org
- port: 65535

The port number is NOT the default mumble port number. If you find an empty server then please check your port number.

When you first join the server it will ask you to choose a username. Please choose something like *FirstnameLastname*. The user name must not have any spaces.

If it's your first time dialing into the call, on the mumble application's configuration section please select "Push to Talk" and/or use a headset to reduce noise on the call.



## Weekly dev calls

The weekly development call is held on the mumble server in the "Weekly Dev Call" channel. Time of the meeting is: 2300 UTC A calendar with the meeting can be found at

[https://calendar.google.com/calendar/embed?](https://calendar.google.com/calendar/embed?src=rgdbom27tb1vlo62kjnjnmt8va4%40group.calendar.google.com)

<src=rgdbom27tb1vlo62kjnjnmt8va4%40group.calendar.google.com> which will show the correct time for your timezone.

The agenda and minutes are normally sent to discuss.ardupilot.org.

## Prefer Typing?

If you prefer text chat, please join us on [gitter](#)

## MAVLink Commands

ArduPilot has adopted a subset of the MAVLink protocol command set. Important links for working with commands are listed below:

- [MAVLINK Common Message Set in HTML and XML \(Protocol Definition\)](#).
- [MavLink Tutorial for Absolute Dummies \(Part-1\)](#) (from [Shyam Balasubramanian](#)).
- [MAVLink Mission Command Messages \(MAV\\_CMD\)](#) (ArduPilot supported subset).
- User-modifiable MAVLink parameters: [Copter Parameters](#), [Plane Parameters](#), [Rover Parameters](#).

Other relevant topics on this wiki include:

## ArduPilot MAVLink Command Package Format

ArduPilot's MAVLink commands are stored in 14 bytes, arranged as follows:

Byte #	Address	Data type	Function
0	0x00	byte	Command ID
1	0x01	byte	Options
2	0x02	byte	Parameter 1
3		long	Parameter 2
4	0x04	..	Parameter 3
5	0x05	..	
6	0x06	..	
7	0x07	long	
8	0x08	..	Parameter 4
9	0x09	..	
10	0x0A	..	
11	0x0B	long	
12	0x0C	..	Parameter 5
13	0x0D	..	
14	0x0E	..	

## Copter Commands in Guided Mode

This article lists the commands that are handled by Copter in GUIDED mode (for example, when writing GCS or Companion Computer apps in [DroneKit](#)). Except where explicitly stated, most of these can also be called in other modes too.

Note

The list is inferred from Copter's [GCS\\_Mavlink.cpp](#) for AC3.3.

### Movement commands

These commands can only be called in GUIDED Mode. They are used for position and velocity control of the vehicle.

[SET\\_POSITION\\_TARGET\\_LOCAL\\_NED](#)

[SET\\_POSITION\\_TARGET\\_GLOBAL\\_INT](#)

## MAV\_CMDs

These MAV\_CMDs can be processed if packaged within a [COMMAND\\_LONG](#) message.

[MAV\\_CMD\\_NAV\\_TAKEOFF](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_NAV\\_LOITER\\_UNLIM](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_NAV\\_RETURN\\_TO\\_LAUNCH](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_NAV\\_LAND](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_CONDITION\\_YAW](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_CHANGE\\_SPEED](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_SET\\_HOME](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_SET\\_ROI](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_MISSION\\_START](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_COMPONENT\\_ARM\\_DISARM](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_SET\\_SERVO](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_REPEAT\\_SERVO](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_SET\\_RELAY](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_REPEAT\\_RELAY](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_FENCE\\_ENABLE](#) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_PARACHUTE](#) (If parachute enabled) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_DO\\_GRIPPER](#) (If gripper enabled) (Copter 3.2.1 or earlier)

[MAV\\_CMD\\_START\\_RX\\_PAIR](#) (Copter 3.3) Starts receiver pairing

[MAV\\_CMD\\_PREFLIGHT\\_CALIBRATION](#) (Copter 3.3)

[MAV\\_CMD\\_PREFLIGHT\\_SET\\_SENSOR\\_OFFSETS](#) (Copter 3.3)

[MAV\\_CMD\\_PREFLIGHT\\_REBOOT\\_SHUTDOWN](#) (Copter 3.3)

[MAV\\_CMD\\_DO\\_MOTOR\\_TEST](#) (Copter 3.3)

[MAV\\_CMD\\_REQUEST\\_AUTOPILOT\\_CAPABILITIES](#) (Copter 3.3)

[MAV\\_CMD\\_GET\\_HOME\\_POSITION](#) (Copter 3.3)

[MAV\\_CMD\\_DO\\_START\\_MAG\\_CAL](#) (Master - not in Copter 3.3)

[MAV\\_CMD\\_DO\\_ACCEPT\\_MAG\\_CAL](#) (Master - not in Copter 3.3)

[MAV\\_CMD\\_DO\\_CANCEL\\_MAG\\_CAL](#) (Master - not in Copter 3.3)

[MAV\\_CMD\\_DO\\_FLIGHTTERMINATION](#) (Copter 3.3) Disarms motors immediately (Copter falls!).

[MAV\\_CMD\\_DO\\_SEND\\_BANNER](#) - No link available (?)

These MAV\_CMD commands can be sent as their own message type (not inside  
`:ref:`COMMAND_LONG`): MAV\_CMD\_DO\_DIGICAM\_CONFIGURE  
<copter:mav_cmd_do_digicam_configure>`

[MAV\\_CMD\\_DO\\_DIGICAM\\_CONTROL](#)

[MAV\\_CMD\\_DO\\_MOUNT\\_CONFIGURE](#)

[MAV\\_CMD\\_DO\\_MOUNT\\_CONTROL](#)

## Other commands

Below are other (non-MAV\_CMD) commands that will be handled by Copter in GUIDED mode.

Note

Most of these commands are not relevant to DroneKit-Python apps or are already provided through the API.

[HEARTBEAT](#)

[SET\\_MODE](#)

[PARAM\\_REQUEST\\_READ](#)

[PARAM\\_REQUEST\\_LIST](#)

[PARAM\\_SET](#)

[MISSION\\_WRITE\\_PARTIAL\\_LIST](#)

[MISSION\\_ITEM](#)

[MISSION\\_REQUEST](#)

[MISSION\\_SET\\_CURRENT](#)

[MISSION\\_REQUEST\\_LIST](#)

[MISSION\\_COUNT](#)

[MISSION\\_CLEAR\\_ALL](#)

[REQUEST\\_DATA\\_STREAM](#)

[GIMBAL\\_REPORT](#)

RC\_CHANNELS\_OVERRIDE

COMMAND\_ACK

HIL\_STATE

RADIO

RADIO\_STATUS

LOG\_REQUEST\_DATA

LOG\_ERASE

LOG\_REQUEST\_LIST

LOG\_REQUEST\_END

SERIAL\_CONTROL

GPS\_INJECT\_DATA

TERRAIN\_DATA

TERRAIN\_CHECK

RALLY\_POINT

RALLY\_FETCH\_POINT

AUTOPILOT\_VERSION\_REQUEST

LED\_CONTROL

ADSB\_VEHICLE

REMOTE\_LOG\_BLOCK\_STATUS

LANDING\_TARGET (Planned for Copter 3.4)

SET\_HOME\_POSITION (Master branch - not in Copter 3.3)

## Command definitions

This section contains information about some immediate commands supported by Copter (Mission Commands are documented in [MAVLink Mission Command Messages \(MAV\\_CMD\)](#)).

Note

Editors: It may make sense to merge the immediate command information for Copter/Plane/Rover as done for [Mission Commands](#) when we have a few more

### **SET\_POSITION\_TARGET\_LOCAL\_NED**

Set vehicle position or velocity setpoint in local frame.

## Note

Starting in Copter 3.3, velocity commands should be resent every second (the vehicle will stop after a few seconds if no command is received). Prior to Copter 3.3 the command was persistent, and would only be interrupted when the next movement command was received.

### Command parameters

Command Field	Description
<code>time_boot_ms</code>	Timestamp in milliseconds since system boot. The rationale for the timestamp in the setpoint is to allow the system to compensate for the transport delay of the setpoint. This allows the system to compensate processing latency.
<code>target_system</code>	System ID
<code>target_component</code>	Component ID
<code>coordinate_frame</code>	Valid options are listed below
<code>type_mask</code>	<p>Bitmask to indicate which dimensions should be ignored by the vehicle (a value of 0b0000000000000000 or 0b0000001000000000 indicates that none of the setpoint dimensions should be ignored). Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7: ax, bit 8: ay, bit 9:</p> <p><b>Note</b></p> <p>At time of writing you <b>must</b> enable all three bits for the position (0b00001111111100) OR all three bits for the velocity (0b0000111111000111). Setting just one bit of the position/velocity or mixing the bits is not supported. The <b>acceleration</b>, <b>yaw</b>, <b>yaw_rate</b> are present in the <a href="#">protocol definition</a> but are not supported by ArduPilot.</p>
<code>x</code>	X Position in specified NED frame in meters
<code>y</code>	y Position in specified NED frame in meters
<code>z</code>	Z Position in specified NED frame in meters (note, altitude is negative in NED)
<code>vx</code>	X velocity in specified NED frame in meter/s
<code>vy</code>	Y velocity in NED frame in meter/s
<code>vz</code>	Z velocity in NED frame in meter/s
<code>afx</code>	X acceleration or force (if bit 10 of type_mask is set) in specified NED frame in meter/s^2 or N
<code>afy</code>	Y acceleration or force (if bit 10 of type_mask is set) in specified NED frame in meter/s^2 or N
<code>afz</code>	Z acceleration or force (if bit 10 of type_mask is set) in specified NED frame in meter/s^2 or N
<code>yaw</code>	yaw setpoint in rad
<code>yaw_rate</code>	yaw rate setpoint in rad/s

## Note

The `co-ordinate frame` information below applies from AC3.3. Prior to AC3.3, the field is ignored and all values are relative to the `MAV_FRAME_LOCAL_NED` frame.

The `co-ordinate frame` field takes the following values:

Frame	Description
<code>MAV_FRAME_LOCAL_NED</code>	Positions are relative to the vehicle's <i>home position</i> in the North, East, Down (NED) frame. Use this to specify a position x metres north, y metres east and (-) z metres above the home position.

	Velocity directions are in the North, East, Down (NED) frame.
<code>MAV_FRAME_LOCAL_OFFSET_NED</code>	Positions are relative to the current vehicle position in the North, East, Down (NED) frame. Use this to specify a position x metres north, y metres east and (-) z metres of the current vehicle position.  Velocity directions are in the North, East, Down (NED) frame.
<code>MAV_FRAME_BODY_OFFSET_NED</code>	Positions are relative to the current vehicle position in a frame based on the vehicle's current heading. Use this to specify a position x metres forward from the current vehicle position, y metres to the right, and z metres down (forward, right and down are "positive" values).  Velocity directions are relative to the current vehicle heading. Use this to specify the speed forward, right and down (or the opposite if you use negative values).
<code>MAV_FRAME_BODY_NED</code>	Positions are relative to the vehicle's <i>home position</i> in the North, East, Down (NED) frame. Use this to specify a position x metres north, y metres east and (-) z metres above the home position.  Velocity directions are relative to the current vehicle heading. Use this to specify the speed forward, right and down (or the opposite if you use negative values).

### Tip

In frames, `_OFFSET` means "relative to vehicle position" while `_LOCAL` is "relative to home position" (these have no impact on *velocity* directions). `_BODY` means that velocity components are relative to the heading of the vehicle rather than the NED frame.

## **SET\_POSITION\_TARGET\_GLOBAL\_INT**

Set vehicle position, velocity and acceleration setpoint in the WGS84 coordinate system.

### Note

Starting in Copter 3.3, velocity commands should be resent every second (the vehicle will stop after a few seconds if no command is received). Prior to Copter 3.3 the command was persistent, and would only be interrupted when the next movement command was received.

The protocol definition is here: [SET\\_POSITION\\_TARGET\\_GLOBAL\\_INT](#).

### Command parameters

Command Field	Description
<code>time_boot_ms</code>	Timestamp in milliseconds since system boot. The rationale for the timestamp in the setpoint is to allow the system to compensate for the transport delay of the setpoint. This allows the system to compensate processing latency.
<code>target_system</code>	System ID

<b>target_component</b>	Component ID
<b>coordinate_frame</b>	Valid options are: MAV_FRAME_GLOBAL_INT, MAV_FRAME_GLOBAL_RELATIVE_ALT_INT
<b>type_mask</b>	<p>Bitmask to indicate which dimensions should be ignored by the vehicle (a value of 0b0000000000000000 or 0b0000010000000000 indicates that none of the setpoint dimensions should be ignored). Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7: ax, bit 8: ay, bit 9:</p> <p><b>Note</b></p> <p>At time of writing you <b>must</b> enable all three bits for the position (0b000011111111000) OR all three bits for the velocity (0b000011111100011). Setting just one bit of the position/velocity or mixing the bits is not supported. The <b>acceleration</b>, <b>yaw</b>, <b>yaw_rate</b> are present in the <a href="#">protocol definition</a> but are not supported by ArduPilot.</p>
<b>lat_int</b>	X Position in WGS84 frame in 1e7 /* meters
<b>lon_int</b>	Y Position in WGS84 frame in 1e7 /* meters
<b>alt</b>	Altitude in meters in AMSL altitude, not WGS84 if absolute or relative, above terrain if GLOBAL_TERRAIN_ALT_INT
<b>vx</b>	X velocity in MAV_FRAME_LOCAL_NED frame in meter/s
<b>vy</b>	Y velocity in MAV_FRAME_LOCAL_NED frame in meter/s
<b>vz</b>	Z velocity in MAV_FRAME_LOCAL_NED frame in meter/s
<b>afx</b>	X acceleration or force (if bit 10 of type_mask is set) in specified MAV_FRAME_LOCAL_NED frame in meter/s^2 or N
<b>afy</b>	Y acceleration or force (if bit 10 of type_mask is set) in specified MAV_FRAME_LOCAL_NED frame in meter/s^2 or N
<b>afz</b>	Z acceleration or force (if bit 10 of type_mask is set) in specified MAV_FRAME_LOCAL_NED frame in meter/s^2 or N
<b>yaw</b>	yaw setpoint in rad
<b>yaw_rate</b>	yaw rate setpoint in rad/s

## SET\_HOME\_POSITION

The position the system will return to and land on. The position is set automatically by the system during the takeoff if it has not been explicitly set by the operator before or after.

### Note

Not in Copter 3.3 (currently in master)

### Command parameters

Command Field	Type	Description
<b>target_system</b>	uint8_t	System ID
<b>latitude</b>	int32_t	Latitude (WGS84), in degrees /* 1E7
<b>longitude</b>	int32_t	Longitude (WGS84), in degrees /* 1E7
<b>altitude</b>	int32_t	Altitude (AMSL), in meters /* 1000 (positive for up)
<b>x</b>	float	Local X position of this position in the local coordinate frame.
<b>y</b>	float	Local Y position of this position in the local coordinate frame
<b>z</b>	float	Local Z position of this position in the local coordinate frame
		World to surface normal and heading transformation of the takeoff position. Used

<code>q</code>	float[4]	to indicate the heading and slope of the ground.
<code>approach_x</code>	float	Local X position of the end of the approach vector. Multicopters should set this position based on their takeoff path. Grass-landing fixed wing aircraft should set it the same way as multicopters. Runway-landing fixed wing aircraft should set it to the opposite direction of the takeoff, assuming the takeoff happened from the threshold / touchdown zone.
<code>approach_y</code>	float	Local Y position of the end of the approach vector. Multicopters should set this position based on their takeoff path. Grass-landing fixed wing aircraft should set it the same way as multicopters. Runway-landing fixed wing aircraft should set it to the opposite direction of the takeoff, assuming the takeoff happened from the threshold / touchdown zone.
<code>approach_z</code>	float	Local Z position of the end of the approach vector. Multicopters should set this position based on their takeoff path. Grass-landing fixed wing aircraft should set it the same way as multicopters. Runway-landing fixed wing aircraft should set it to the opposite direction of the takeoff, assuming the takeoff happened from the threshold / touchdown zone.

The protocol definition for this command is here: [SET\\_HOME\\_POSITION](#)

## Plane Commands in Guided Mode

This article lists the commands that are handled by Plane in GUIDED mode (for example, when writing GCS or Companion Computer apps in [DroneKit](#)). Except where explicitly stated, most of these can also be called in other modes too.

### Note

The list is inferred from Plane's [GCS\\_Mavlink.cpp](#)

### Movement commands

For movement, Plane uses:

[MAV\\_CMD\\_NAV\\_WAYPOINT](#) message encoded with the "current" parameter set to "2" to indicate that it is a guided mode "goto" message.

[MAV\\_CMD\\_CONDITION\\_CHANGE\\_ALT](#) message encoded with the "current" parameter set to "3" to indicate that it is a guided mode "altitude change" message.

### MAV\_CMDS

These MAV\_CMDS can be processed if packaged within a [COMMAND\\_LONG](#) message.

[MAV\\_CMD\\_NAV\\_LOITER\\_UNLIM](#)

[MAV\\_CMD\\_NAV\\_RETURN\\_TO\\_LAUNCH](#)

[MAV\\_CMD\\_DO\\_SET\\_ROI](#)

[MAV\\_CMD\\_MISSION\\_START](#)

[MAV\\_CMD\\_COMPONENT\\_ARM\\_DISARM](#)

### Todo

[MAV\\_CMD\\_MISSION\\_START](#) and [MAV\\_CMD\\_COMPONENT\\_ARM\\_DISARM](#) not implemented as auto commands (or at least not when I did the master doc. Need to confirm that they aren't AUTO commands, that they are guided commands, and if so then document either here or in the command list.

MAV\_CMD\_DO\_SET\_SERVO  
MAV\_CMD\_DO\_REPEAT\_SERVO  
MAV\_CMD\_DO\_SET\_RELAY  
MAV\_CMD\_DO\_REPEAT\_RELAY  
MAV\_CMD\_DO\_FENCE\_ENABLE  
MAV\_CMD\_DO\_SET\_HOME  
MAV\_CMD\_START\_RX\_PAIR  
MAV\_CMD\_PREFLIGHT\_CALIBRATION  
MAV\_CMD\_PREFLIGHT\_SET\_SENSOR\_OFFSETS  
MAV\_CMD\_DO\_SET\_MODE  
MAV\_CMD\_PREFLIGHT\_REBOOT\_SHUTDOWN  
MAV\_CMD\_DO\_LAND\_START  
MAV\_CMD\_DO\_GO\_AROUND  
MAV\_CMD\_REQUEST\_AUTOPILOT\_CAPABILITIES  
**MAV\_CMD\_DO\_AUTOTUNE\_ENABLE**

These MAV\_CMD commands can be sent as their own message type (not inside :ref:`COMMAND\_LONG`): `MAV_CMD_DO_DIGICAM_CONFIGURE <plane:mav_cmd_do_digicam_configure>`

**MAV\_CMD\_DO\_DIGICAM\_CONTROL**  
MAV\_CMD\_DO\_MOUNT\_CONFIGURE  
**MAV\_CMD\_DO\_MOUNT\_CONTROL**

## Other commands

Below are other (non-MAV\_CMD) commands that will be handled by Plane in GUIDED mode.

Note

Most of these commands are not relevant to DroneKit-Python apps or are already provided through the API.

**SET\_MODE**

**MISSION\_REQUEST\_LIST**

**MISSION\_REQUEST**

MISSION\_ACK:

**PARAM\_REQUEST\_LIST**

PARAM\_REQUEST\_READ  
MISSION\_CLEAR\_ALL  
MISSION\_SET\_CURRENT  
MISSION\_COUNT  
MISSION\_WRITE\_PARTIAL\_LIST  
MISSION\_ITEM  
MAVLINK\_MSG\_ID\_FENCE\_POINT  
MAVLINK\_MSG\_ID\_FENCE\_FETCH\_POINT  
RALLY\_POINT  
RALLY\_FETCH\_POINT  
PARAM\_SET  
GIMBAL\_REPORT  
RC\_CHANNELS\_OVERRIDE  
HEARTBEAT  
HIL\_STATE  
RADIO  
RADIO\_STATUS  
LOG\_REQUEST\_DATA  
LOG\_ERASE  
LOG\_REQUEST\_LIST  
LOG\_REQUEST\_END  
SERIAL\_CONTROL  
GPS\_INJECT\_DATA  
TERRAIN\_DATA  
TERRAIN\_CHECK  
AUTOPilot\_VERSION\_REQUEST  
REQUEST\_DATA\_STREAM

## MAVLink Routing in ArduPilot

This topic explains how MAVLink routing is handled in ArduPilot (AC3.3 and later).

## Overview

A MAVLINK network is made up of *systems* (vehicles, GCS, antenna trackers etc.) which are themselves made up of components (Autopilot, camera system, etc.). The protocol defines two ids that can be specified in messages to control routing of the command to a required system and component:

- `target_system`: System which should execute the command
- `target_component`: Component which should execute the command

By default a GCS will typically have a system id of 255, and vehicles have an ID of 1. In a system with multiple vehicles (or including an AntennaTracker), each system should be given a unique ID (`SYSID_THISMAV`). A value of 0 for the `target_system` or `target_component` is considered a broadcast ID, and will be sent to all systems in the network/components on the target system.

The routing on ArduPilot systems works in the following way:

- All received MAVLINK messages are checked by the `MAVLINK_routing` class.
- The class extracts the source system id (aka `sysid`) and component id (aka `compid`) and builds up a routing array which maps the channel (i.e. USB port, Telem1, Telem2) to the `<sysid, compid>` pair.
- The class also extracts the target system id (`target_system`) and component id (`target_component`) and if these don't match the vehicle's `<sysid, compid>` the messages are forwarded to the appropriate channel using the array above.
- Messages that don't have a target `<sysid, compid>` are processed by the vehicle and then forwarded to each known system/component.

## Detailed theory of MAVLink routing

This information below is reproduced from the routing code in-source comments ([/libraries/GCS\\_MAVLink/MAVLINK\\_routing.cpp](#))

1. When a flight controller receives a message it should process it locally if any of these conditions hold:
  1. It has no `target_system` field
  2. It has a `target_system` of zero
  3. It has the flight controllers target system and has no `target_component` field
  4. It has the flight controllers target system and has the flight controllers `target_component`
  5. It has the flight controllers target system and the flight controller has not seen any messages on any of its links from a system that has the messages `target_system` / `target_component` combination
2. When a flight controller receives a message it should forward it onto another different link if any of these conditions hold for that link:
  1. It has no `target_system` field
  2. It has a `target_system` of zero
  3. It does not have the flight controllers `target_system` and the flight controller has seen a message from the messages `target_system` on the link
  4. It has the flight controllers `target_system` and has a `target_component` field and the flight controllers has seen a message from the `target_system` / `target_component` combination on the link

### Note

This proposal assumes that ground stations will not send command packets to a non-broadcast destination (sysid/compid combination) until they have received at least one package from that destination over the link. This is essential to prevent a flight controller from acting on a message that is not meant for



it. For example, a `PARAM_SET` cannot be sent to a specific `<sysid/compid>` combination until the GCS has seen a packet from that `<sysid/compid>` combination on the link.

The GCS must also reset what`<sysid/compid>` combinations it has seen on a link when it sees a `SYSTEM_TIME` message with a decrease in `time_boot_ms` from a particular `<sysid/compid>`. That is essential to detect a reset of the flight controller, which implies a reset of its routing table.

## Companion Computers

Companion Computers can be used to interface and communicate with ArduPilot on a flight controller using the MAVLink protocol. By doing this your companion computer gets all the MAVLink data produced by the autopilot (including GPS data) and can use it to make intelligent decisions during flight. For example, “take a photo when the vehicle is at these GPS co-ordinates”.

The possibilities of companion computers are really only limited by your imagination.

Related topics on this wiki include:

## DroneKit Tutorial

DroneKit-Python allows you to control ArduPilot using the Python programming language.

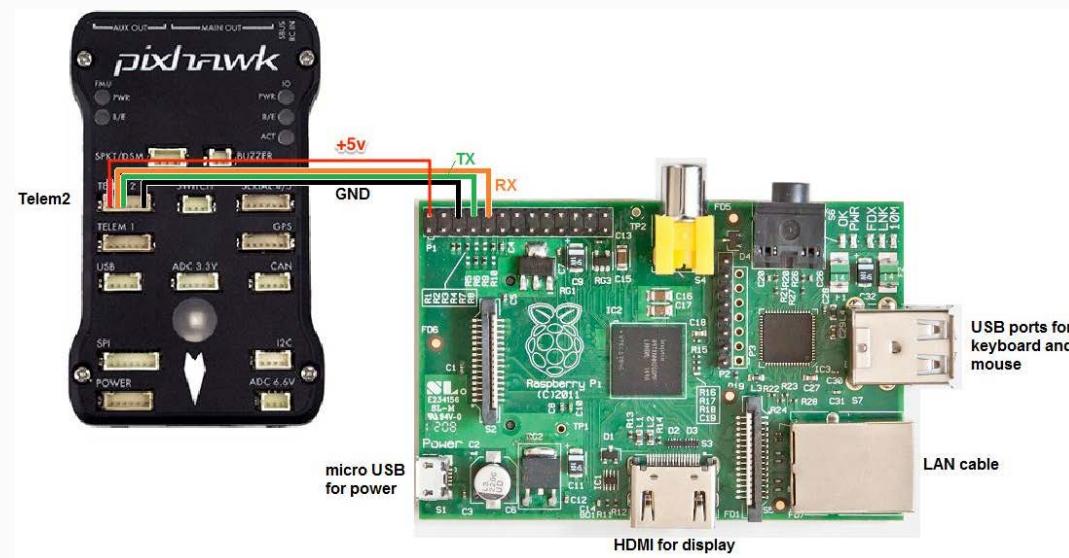
The [official DroneKit Python documentation](#) contains a [quick start guide](#).

There is also a video below showing how to setup Dronekit for SITL/MAVProxy on Linux.

## Communicating with Raspberry Pi via MAVLink

This page explains how to connect and configure a Raspberry Pi (RPI) so that it is able to communicate with a Pixhawk flight controller using the MAVLink protocol over a serial connection. This can be used to perform additional tasks such as image recognition which simply cannot be done by the Pixhawk due to the memory requirements for storing images.

### Connecting the Pixhawk and RPi



Connect the Pixhawk's TELEM2 port to the RPi's Ground, TX and RX pins as shown in the image above.

More details on the individual RPi's pin functions can be found [here](#).

The RPi can be powered by connecting the red V+ cable to the +5V pin (as shown above) **or** from USB in (for example, using a separate 5V BEC hooked up to the USB power).

### Tip

Powering via USB is recommended as it is typically safer - because the input is regulated. If powering via USB, do not also connect the +5V pin as shown (still connect common ground).

## Connecting to RPi with an SSH/Telnet client

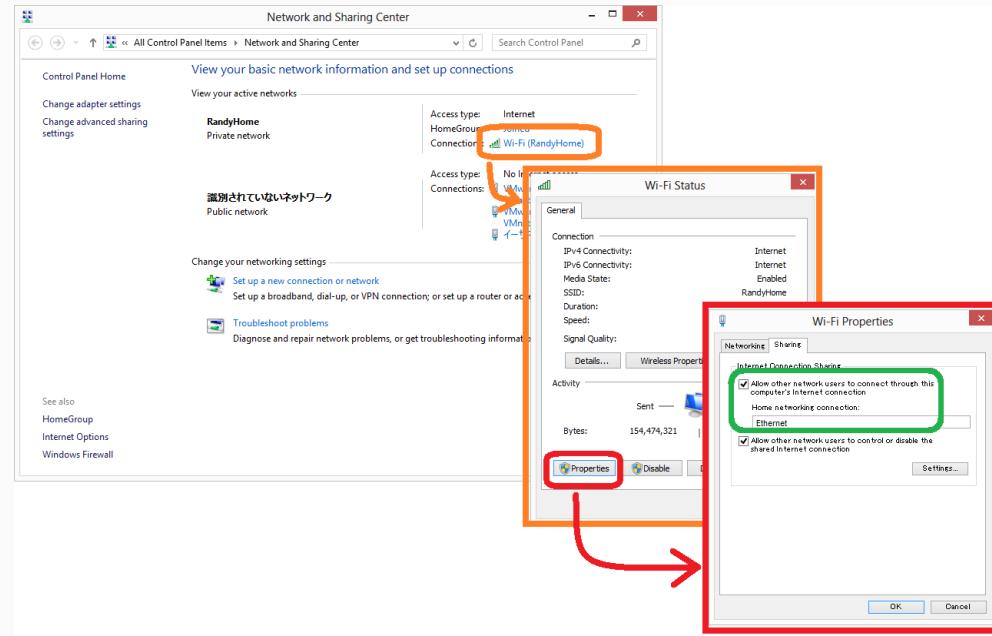
### Note

These steps assume that you have already [set-up your RPi](#) so that it is running Raspbian.

To avoid the requirement to plug a keyboard, mouse and HDMI screen into your RPi it is convenient to be able to connect from your Desktop/Laptop computer to your RPi using an SSH/Telnet client such as [PuTTY](#).

1. Connect the RPi to your local network by one of the following methods:

- Connecting an Ethernet LAN cable from the RPi board to your Ethernet router, or
- [Use a USB dongle to connect your RPi to the local wireless network](#), or
- Connect the Ethernet LAN cable between the RPi and your desktop/laptop computer. Open the control panel's Network and Sharing Center, click on the network connection through which your desktop/laptop is connected to the internet, select properties and then in the sharing tab, select "Allow other networks to connect through this computer's Internet connection"

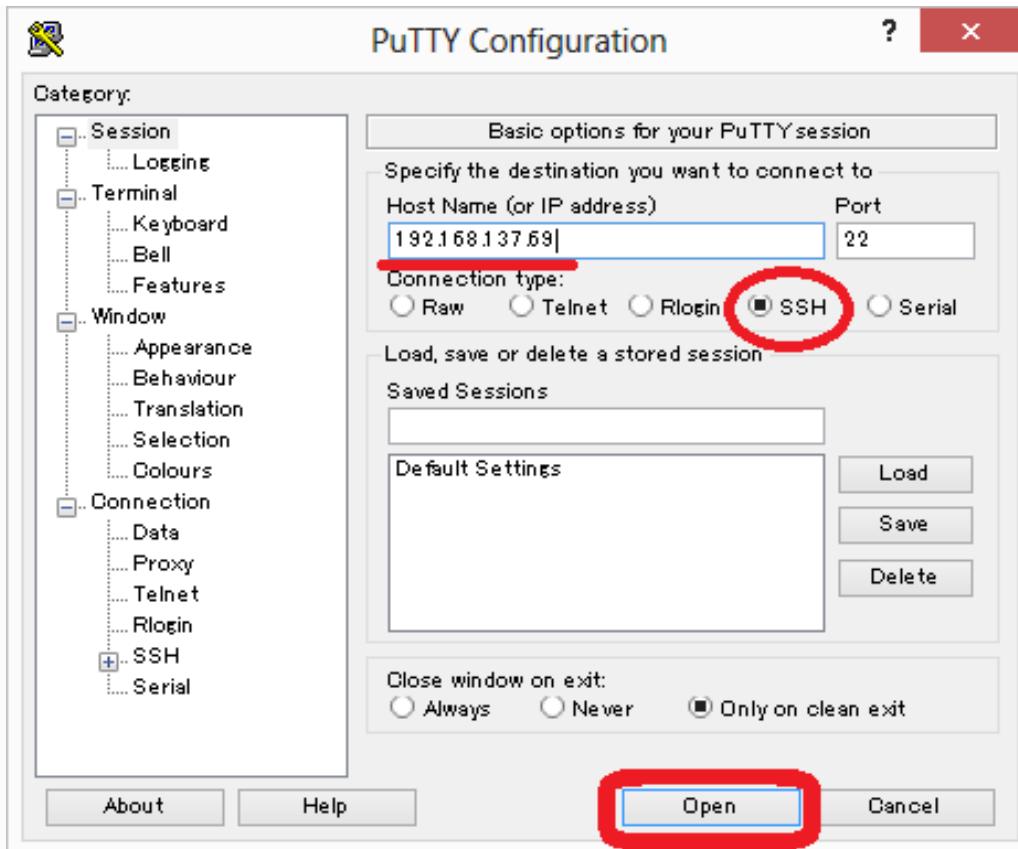


2. Determine the RPi's IP address:

- If you have access to the RPi's terminal screen (i.e. you have a screen, keyboard, mouse connected) you can use the /sbin/ifconfig command.

- If your Ethernet router has a web interface it may show you the IP address of all connected devices

3. Connect with [Putty](#):



If all goes well you should be presented with the regular login prompt to which you can enter the username/password which defaults to pi/raspberry

## Install the required packages on the Raspberry Pi

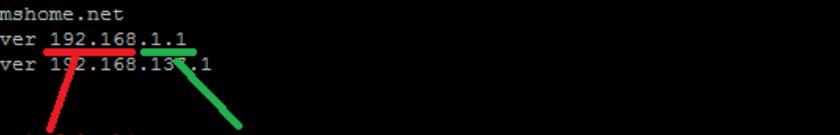
Log into the RPi board (default username password is pi/raspberry) and check that its connection to the internet is working:

```
ping google.com
```

OR

```
ping 173.194.126.196
```

If the first fails but the second succeeds then there is a problem with the DNS server that your RPi is attempting to use. Please edit the /etc/resolv.conf file and add the IP address of a nearby DNS server. During the creation of this wiki, the first two parts of the desktop machine's IP address plus ".1.1" worked. To stop other processes from later updating this file you may wish to run the `chattr +i /etc/resolv.conf` command (this can be undone later with `chattr -i /etc/resolv.conf`). That sets the "immutable" bit on resolv.conf to prevent other software from updating it.



```
pi@raspberrypi: ~
domain mshome.net
search mshome.net
nameserver 192.168.1.1
nameserver 192.168.137.1
-
1st part of desktop PC's IP address .1.1 for the last two digits
/etc/resolv.conf" [readonly] 4 lines, 84 characters
```

After the internet connection is confirmed to be working install these packages

```
sudo apt-get update      #Update the list of packages in the software center  
sudo apt-get install screen python-wxgtk2.8 python-matplotlib python-opencv python-pip python-numpy  
python-dev libxml2-dev libxslt-dev  
sudo pip install pymavlink  
sudo pip install mavproxy
```

## Note

The packages are mostly the same as when setting up SITL. Reply 'y' when prompted re additional disk space.

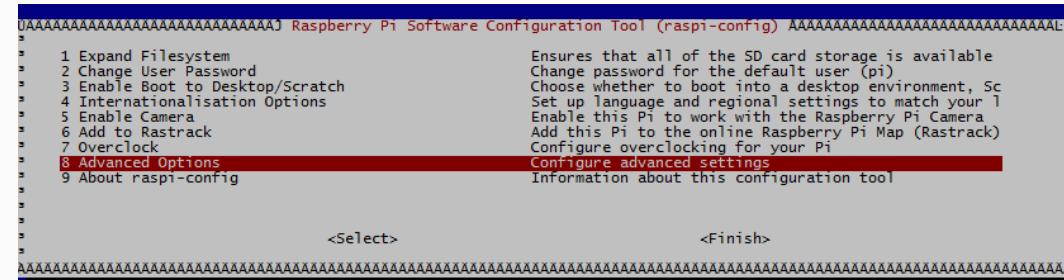
### **Disable the OS control of the serial port**

Use the Raspberry Pi configuration utility for this

### Type

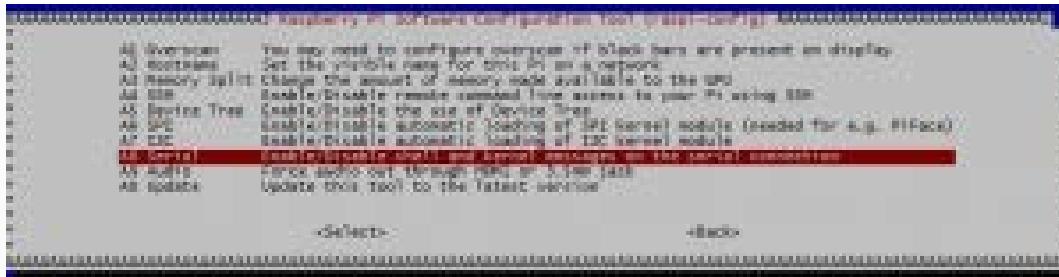
```
sudo raspi-config
```

And in the utility, select “Advanced Options”



## **RasPiConfiguration Utility: Serial Settings: Advanced Options**

And then “Serial” to disable OS use of the serial connection



Reboot the Raspberry Pi when you are done.

## Testing the connection

To test the RPi and Pixhawk are able to communicate with each other first ensure the RPi and Pixhawk are powered, then in a console on the RPi type:

```
sudo -s  
mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --aircraft MyCopter
```

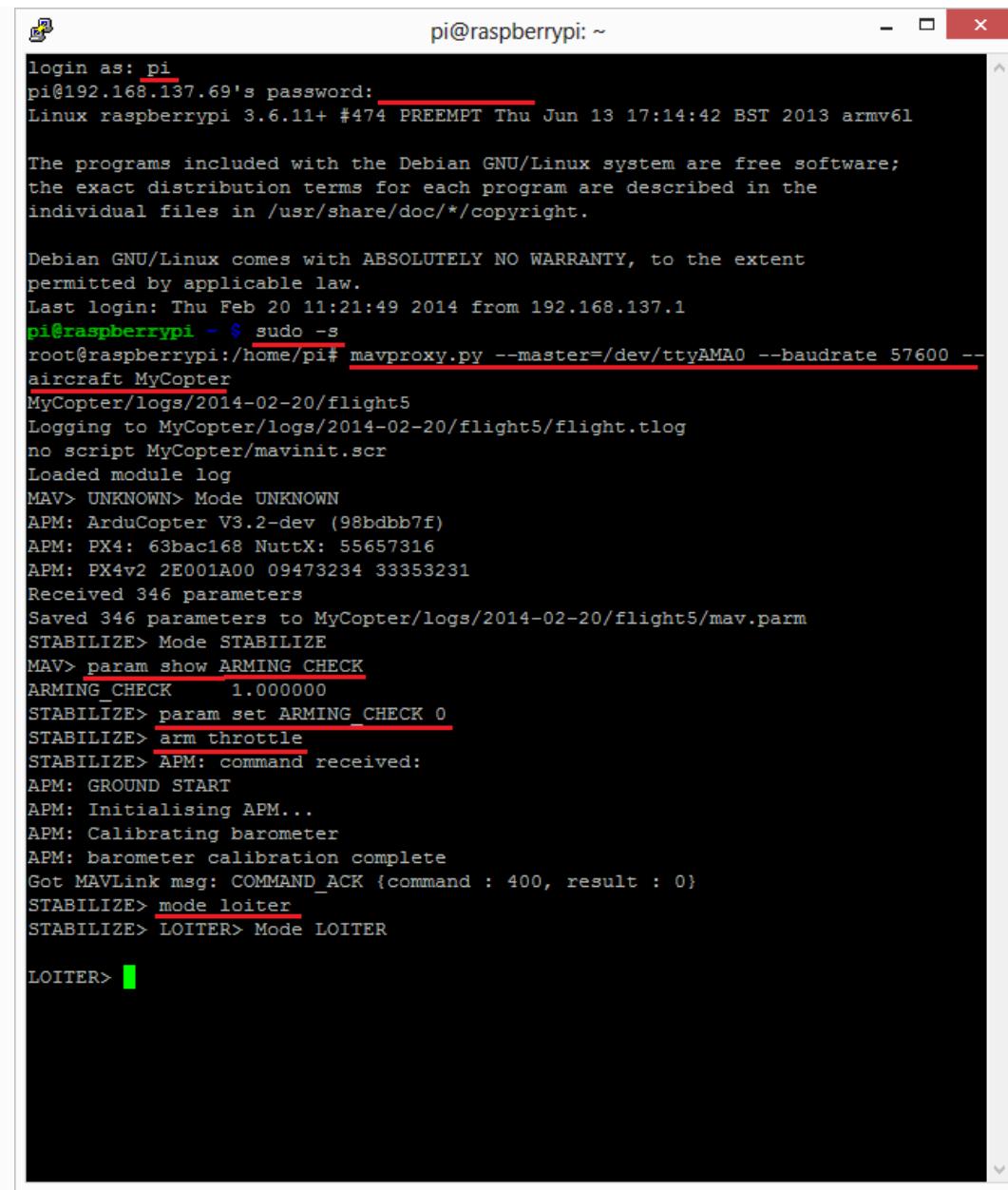
## Note

On newer versions of Raspberry Pi 3 the uart serial connection may be disabled by default. In order to enable serial connection on the Raspberry Pi edit `/boot/config.txt` and set `enable_uart=1`. The build-in serial port is `/dev/ttys0`.

Once MAVProxy has started you should be able to type in the following command to display the

**ARMING\_CHECK** parameters value

```
param show ARMING_CHECK  
param set ARMING_CHECK 0  
arm throttle
```



```

pi@raspberrypi: ~
login as: pi
pi@192.168.137.69's password:
Linux raspberrypi 3.6.11+ #474 PREEMPT Thu Jun 13 17:14:42 BST 2013 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Feb 20 11:21:49 2014 from 192.168.137.1
pi@raspberrypi ~ $ sudo -s
root@raspberrypi:/home/pi# mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --
aircraft MyCopter
MyCopter/logs/2014-02-20/flight5
Logging to MyCopter/logs/2014-02-20/flight5/flight.tlog
no script MyCopter/mavinit.scr
Loaded module log
MAV> UNKNOWN> Mode UNKNOWN
APM: ArduCopter V3.2-dev (98bdbb7f)
APM: PX4: 63bac168 NuttX: 55657316
APM: PX4v2 2E001A00 09473234 33353231
Received 346 parameters
Saved 346 parameters to MyCopter/logs/2014-02-20/flight5/mav.parm
STABILIZE> Mode STABILIZE
MAV> param show ARMING_CHECK
ARMING_CHECK 1.000000
STABILIZE> param set ARMING_CHECK 0
STABILIZE> arm throttle
STABILIZE> APM: command received:
APM: GROUND START
APM: Initialising APM...
APM: Calibrating barometer
APM: barometer calibration complete
Got MAVLink msg: COMMAND_ACK {command : 400, result : 0}
STABILIZE> mode loiter
STABILIZE> LOITER> Mode LOITER

LOITER> 

```

#### Note

If you get an error about not being able to find log files or if this example otherwise doesn't run properly, make sure that you haven't accidentally assigned these files to another username, such as Root.

Entering the following at the Linux command line will ensure that all files belong to the standard Pi login account:

```
sudo chown -R pi /home/pi
```

#### Configure MAVProxy to always run

To setup MAVProxy to start whenever the RPi is restarted open a terminal window and edit the **/etc/rc.local** file, adding the following lines just before the final "exit 0" line:

```
(  
date  
echo $PATH  
PATH=$PATH:/bin:/sbin:/usr/bin:/usr/local/bin  
export PATH
```

```
cd /home/pi
screen -d -m -s /bin/bash mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --aircraft MyCopter
) > /tmp/rc.log 2>&1
exit 0
```

Whenever the RPi connects to the Pixhawk, three files will be created in the /home/pi/MyCopter/logs/YYYY-MM-DD directory:

- **mav.parm** : a text file with all the parameter values from the Pixhawk
- **flight.tlog** : a telemetry log including the vehicles altitude, attitude, etc which can be opened using the mission planner (and a number of other tools)
- **flight.tlog.raw** : all the data in the .tlog mentioned above plus any other serial data received from the Pixhawk which might include non-MAVLink formatted messages like startup strings or debug output

If you wish to connect to the MAVProxy application that has been automatically started you can log into the RPi and type:

```
sudo screen -x
```

To learn more about using MAVProxy please read the [MAVProxy documentation](#).

It is also worth noting that MAVProxy can do a lot more than just provide access to your Pixhawk. By writing python extension modules for MAVProxy you can add sophisticated autonomous behaviour to your vehicle. A MAVProxy module has access to all of the sensor information that your Pixhawk has, and can control all aspects of the flight. To get started with MAVProxy modules please have a look at the [existing modules](#) in the MAVProxy source code.

## Installing DroneKit on RPi

Tip

The most up-to-date instructions for [Installing DroneKit](#) on Linux are in the DroneKit-Python documentation. This information is a summary, and might go out of date.

To install DroneKit-Python dependencies (most of which will already be present from when you installed MAVProxy) and set DroneKit to load when MAVProxy starts:

```
sudo apt-get install python-pip python-dev python-numpy python-opencv python-serial python-pyparsing
python-wxgtk2.8 libxml2-dev libxslt-dev
sudo pip install droneapi
echo "module load droneapi.module.api" >> ~/.mavinit.scr
```

Then open the MAVProxy terminal in the location where your DroneKit script is located and start an example:

```
MANUAL> api start vehicle_state.py
```

Note

If you get a warning that droneapi module has not loaded, you can do so manually in MAVProxy:

```
MANUAL> module load droneapi.module.api
```

## Connecting with the Mission Planner

The Pixhawk will respond to MAVLink commands received through Telemetry 1 and Telemetry 2 ports (see image at top of this page) meaning that both the RPi and the regular ground station (i.e. Mission planner, etc) can be connected. In addition it is possible to connect the Mission Planner to the MAVProxy application running on the RPi [similar to how it is done for SITL](#).

Primarily this means adding an `--out <ipaddress>:14550` to the MAVProxy startup command with the being the address of the PC running the mission planner. On windows the `ipconfig` can be used to determine that IP address. On the computer used to write this wiki page the MAVProxy command became:

```
mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --out 192.168.137.1:14550 --aircraft MyCopter
```

Connecting with the mission planner is shown below:



## Example projects

### [FPV with raspberry Pi](#)

Can't get it to work? Try posting your question in the [APM Forum's APM Code section](#).

## Making a Mavlink WiFi bridge using the Raspberry Pi

### Note

This document is still a work in progress

[Overview](#)

This page will show you how to setup a Raspberry Pi (RPi) as a gateway to a 3DR Telemetry Radio. This will allow you to connect a computer or tablet via WiFi to the Raspberry Pi, and the Raspberry Pi will in turn forward the communication to a drone through a telemetry link.



To accomplish this tutorial you will need the following items:

- Raspberry Pi with SD Card
- A compatible Wifi dongle (not all dongles support AP mode)
- 3DR Telemetry Radio
- A vehicle to connect to

If you don't want to mess around all the configuration steps described here you can also just go to the bottom of the page and download an SD Card image that you can load as described in the section [Getting the Raspberry Pi up and running with Raspbian](#). Just use the image provided at the bottom instead of the image from the Raspberri Pi Website.

#### [Getting the Raspberry Pi up and running with Raspbian](#)

First we need to get the Raspberry Pi up and running with Debian. This is accomplished by downloading the SD Card image from the Raspberry Pi website, and loading it into an SD Card.

1. Go to the [Raspberry Pi downloads page](#) and download the latest raspbian image (as of this writing: [2014-12-24-wheezy-raspbian.zip](#))
2. Follow the guide specific to your operating system to prepare the SD Card with the image you downloaded:
  - [Linux](#)
  - [Windows](#)
  - [OS X](#)
3. Once you have the SD Card ready insert it into the RPi and boot it up
4. The first time you boot the RPi it will load the Raspberry Pi Configuration Utility, setup the following:
  - Expand the file system (This allows you to use all of the free space on the SD Card)
  - Set your timezone
  - Make sure the boot option is set to boot to console
  - Enable SSH under "Advanced Options"
  - Finish the setup and reboot

You now have the Raspberry Pi ready to setup MavProxy and to set it up as an access point so you can use it to create a local WiFi network. To avoid having to continue using a keyboard and monitor it is convenient to connect to the RPi using an SSH client. You can follow the steps on the page [Connecting to RPi with an SSH/Telnet client](#) in order to achieve this.

#### [Setting up the Raspberry Pi as an access point](#)

Now we need to setup the the software that will allow the RPi to act as an access point. For the following steps make sure you have access to the internet from Raspberry Pi, this is needed in order to download the software packages. The easiest way to achieve this is to connect the RPi to an Ethernet connection.

#### [Installing hostapd and a dhcp server](#)

Now connect to the RPi through an SSH connection and type the following commands:

```
sudo apt-get update
sudo apt-get install hostapd isc-dhcp-server
```

These commands install the software needed to setup the RPi as an access point, and to allow it to assign IP addresses to the computers that connect to it (DHCP Server). After you run these commands you may see the following lines in the output:

```
[FAIL] Starting ISC DHCP server: dhcpcd[....] check syslog for diagnostics. ... failed!
failed!
```

Don't worry, this does not affect the functionality.

Now we need to configure these two packages, first we will configure the DHCP Server

#### Configure the DHCP Server

The following steps will guide you through the configuration of the DHCP server. Next we need to edit the file **/etc/dhcp/dhcpd.conf**, this file allows the computers connected through WiFi to automatically get IP Addresses, DNS, and other information to allow them to connect through the RPi.

```
sudo nano /etc/dhcp/dhcpd.conf
```

Find the following lines and comment them out by adding a # in the beginning of the line:

```
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;
```

Make them so they look like this:

```
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;
```

In the same file, find the following lines and uncomment the “authoritative” line by removing the # at the beginning of the line:

```
# If this DHCP server is the official DHCP server for the Local
# network, the authoritative directive should be uncommented.
#authoritative;
```

Make the line so it looks like this:

```
# If this DHCP server is the official DHCP server for the Local
# network, the authoritative directive should be uncommented.
authoritative;
```

At the bottom of this same file, add the following lines:

```
subnet 192.168.42.0 netmask 255.255.255.0 {
    range 192.168.42.10 192.168.42.50;
    option broadcast-address 192.168.42.255;
    option routers 192.168.42.1;
    default-lease-time 600;
    max-lease-time 7200;
    option domain-name "local";
```

```
        option domain-name-servers 8.8.8.8, 8.8.4.4;
    }
```

These lines configure the network address DNS servers and gateway information that the RPi will assign to the clients connecting to it. If you will use the RPi to bridge to another network through the ethernet connection, it is important that this address range does not conflict with the network the RPi is connected to through the ethernet connection. In other words, if your local network uses the address range 192.168.42.xx you need to select a different address range. Also the router address 192.168.42.1 will be the ip address of the RPi (we will set that up later on).

Save the file by typing in **Control-X** then **Y** then **return**

Now we will edit the file **/etc/default/isc-dhcp-server** in order to tell the server on what network interface it will be active. Type the following command:

```
sudo nano /etc/default/isc-dhcp-server
```

Find the line that says

```
interfaces=""
```

and change it to:

```
interfaces="wlan0"
```

Save the file by typing in **Control-X** then **Y** then **return**

Setup the wlan0 interface as a static ip address

We now have to setup the wireless interface on the RPi to have its own fixed IP address and set it up to take incoming connections, type the following commands:

```
sudo ifdown wlan0
sudo nano /etc/network/interfaces
```

change the file so it matches the following:

```
auto lo

iface lo inet loopback

iface eth0 inet dhcp

allow-hotplug wlan0

iface wlan0 inet static
    address 192.168.42.1
    netmask 255.255.255.0

#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
#iface default inet dhcp
```

Save the file by typing in **Control-X** then **Y** then **return**

This sets up a static IP address (192.168.42.1) to the wireless lan interface (wlan0). This configuration however will not take effect until the next reboot, so if you want to immediately assign this address you need to type the following:

```
sudo ifconfig wlan0 192.168.42.1
```

### Configure the Access Point Details¶

The next steps will set up our wireless network with a name and password, and will configure the hardware adapter to be used in access point (AP) mode. It is important to note that not all WiFi dongles support AP mode, please make sure that the dongle you are using supports this mode. *This has only been tested using the WiFi dongle sold by Adafruit. Check this with Craig*

The configuration file we need to change is **/etc/hostapd/hostapd.conf**

Type the following command:

```
sudo nano /etc/hostapd/hostapd.conf
```

This will create a new file. If there is already a file with that name already, change the contents to the lines below, otherwise add the following lines to the file:

```
interface=wlan0
driver=rtl871xdrv
ssid=MavStation
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=MavLink_1
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Save the file by typing in **Control-X** then **Y** then **return**

This is telling the access point software to use the **wlan0** interface, and to setup a network called **MavStation** with the pass phrase **MavLink1**. Another important part of this file is the line:

```
driver=rtl871xdrv
```

If you are using a different dongle than the one we are using you may need to change this line to use a driver suitable for your dongle, you may want to try:

```
driver=nl80211
```

Make sure the file has no extra spaces or tabs at the beginning and ends of the lines, this file is very sensitive to this.

Now we must tell the software to use the configuration file we just created. To do this we need to modify the file **/etc/default/hostapd**

Type the following command:

```
sudo nano /etc/default/hostapd
```

Find the line that contains **#DAEMON\_CONF=""** and change it to the following:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Save the file by typing in **Control-X** then **Y** then **return**

Configure Network Address Translation (NAT)

Setting up NAT allows the WiFi clients of the RPi to have their data tunneled through the ethernet connection on the RPi. To do this type the following command:

```
sudo nano /etc/sysctl.conf
```

Find the lines:

```
# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1
```

and uncomment it like this:

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

This change will not be applied until the next boot, so to apply it immediately run the following command:

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

Now run the following commands to setup the routing tables between the wireless lan interface and the ethernet port:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

To check your changes to the tables you can use the following commands:

```
sudo iptables -t nat -S
sudo iptables -S
```

In order to restore this changes after boot we need to save the configuration to a file so that we can use that later to restore the configuration. Type the following:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

We now need to update the interface file again, type the following command:

```
sudo nano /etc/network/interfaces
```

And add the following line to the end of the file:

```
up iptables-restore < /etc/iptables.ipv4.nat
```

The complete file should now look like this:

```

auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0

iface wlan0 inet static
    address 192.168.42.1
    netmask 255.255.255.0

#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
#iface default inet dhcp

up iptables-restore < /etc/iptables.ipv4.nat

```

Save the file by typing in **Control-X** then **Y** then **return**

We are now almost ready to run the access point software. Before we do that though we need to update it to a version that supports our network adapter.

#### Update hostapd¶

The hostapd version installed by apt-get does not fully support the WiFi dongle we are using so we need to update it to a later version. Lets get the new version of hostapd by typing the following command:

```
wget http://adafruit-download.s3.amazonaws.com/adafruit_hostapd_14128.zip
```

Now lets unzip the files, swap them with the old version, and fix the permissions so we are able to run the software:

```

unzip adafruit_hostapd_14128.zip
sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.ORIG
sudo mv hostapd /usr/sbin
sudo chmod 755 /usr/sbin/hostapd

```

#### Setting up a daemon¶

With everything installed and ready to go, we now have to set the system up as a program that will start when the system boots up. This is called a **daemon**. Type the following commands:

```

sudo service hostapd start
sudo service isc-dhcp-server start

```

You should now see the following output if everything runs well:

```

pi@MavStation ~ $ sudo service hostapd start
[ ok ] Starting advanced IEEE 802.11 management: hostapd.
pi@MavStation ~ $ sudo service isc-dhcp-server start
[ ok ] Starting ISC DHCP server: dhcpcd.

```

Now in order to set the services up so they run everytime the RPi boots, type the following:

```

sudo update-rc.d hostapd enable
sudo update-rc.d isc-dhcp-server enable

```

One last step is to remove WPA Supplicant so it does not interfere with the Access Point, type the following commands:

```
sudo mv /usr/share/dbus-1/system-services/fi.epitest.hostap.WPASupplicant.service ~/
```

And finally reboot your RPi by typing the following:

```
sudo reboot
```

Your system is now setup as an access point. You should now see a WiFi Network called **MavStation** and you should be able to connect by using the pass phrase **MavLink\_1** (if you chose to keep the same name and pass phrase)

#### Important Considerations!

You must make sure that the RPi is receiving enough power to handle both the WiFi Dongle and the Ethernet connection in case you want to use it as a router to another network. I had success powering it up with a 2A power supply.

If you do not have enough power you may be able to connect to it, but when you try to browse a web page it may drop the connection and kill the network interfaces. Check /var/log/syslog for this type of entries:

```
Dec 27 12:37:17 MavStation kernel: [ 261.984400] ERROR::dwc_otg_hcd_urb_enqueue:505: Not connected
Dec 27 12:37:18 MavStation dhclient: receive_packet failed on eth0: Network is down
Dec 27 12:37:22 MavStation ntpd[2081]: Deleting interface #2 eth0, 10.0.1.21#123, interface stats:
received=11, sent=16, dropped=0, active_time=237 secs
```

Also if you are able to see the Wireless network but can't connect to it, that may also mean that the WiFi channel may be conflicting with another network. Try changing the channel on the file **/etc/hostapd/hostapd.conf**.

```
channel=x
```

Finally make sure that your WiFi dongle supports Access Point mode.

#### Installing and configuring MavProxy

With the RPi now working as an access point, we now need to configure it to connect to a drone. To accomplish this we will install **MavProxy**, a minimalist but full featured ground control station.

#### Tip

The official instructions to [install MAVProxy on Linux](#) are here.

First install some other modules that are needed. Type the following commands:

```
sudo apt-get update
sudo apt-get install python-opencv python-wxgtk python-pip python-dev
```

Then use *pip* to install MavProxy and all its dependencies

```
sudo pip install MAVProxy
```

In order to allow serial connections to the RPi we need to disable the console and login prompt on the serial port. To do this we have to edit the file **/etc/inittab**. Type the following:

```
sudo nano /etc/inittab
```

Now go to the bottom of the file and look for the following lines:

```
#Spawn a getty on Raspberry Pi serial Line
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

and comment out the line like this:

```
#Spawn a getty on Raspberry Pi serial Line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Save the file by typing in **Control-X** then **Y** then **return**

Now we should be ready to test the connection to the drone.

Connecting a 3DR Telemetry Radio to a Raspberry Pi

Tip

The instructions here show how to connect the RPi using the 3DR Radio via a serial port. It is far simpler to connect to the radio via its USB port. The [configuration](#) is the same except that you need to specify

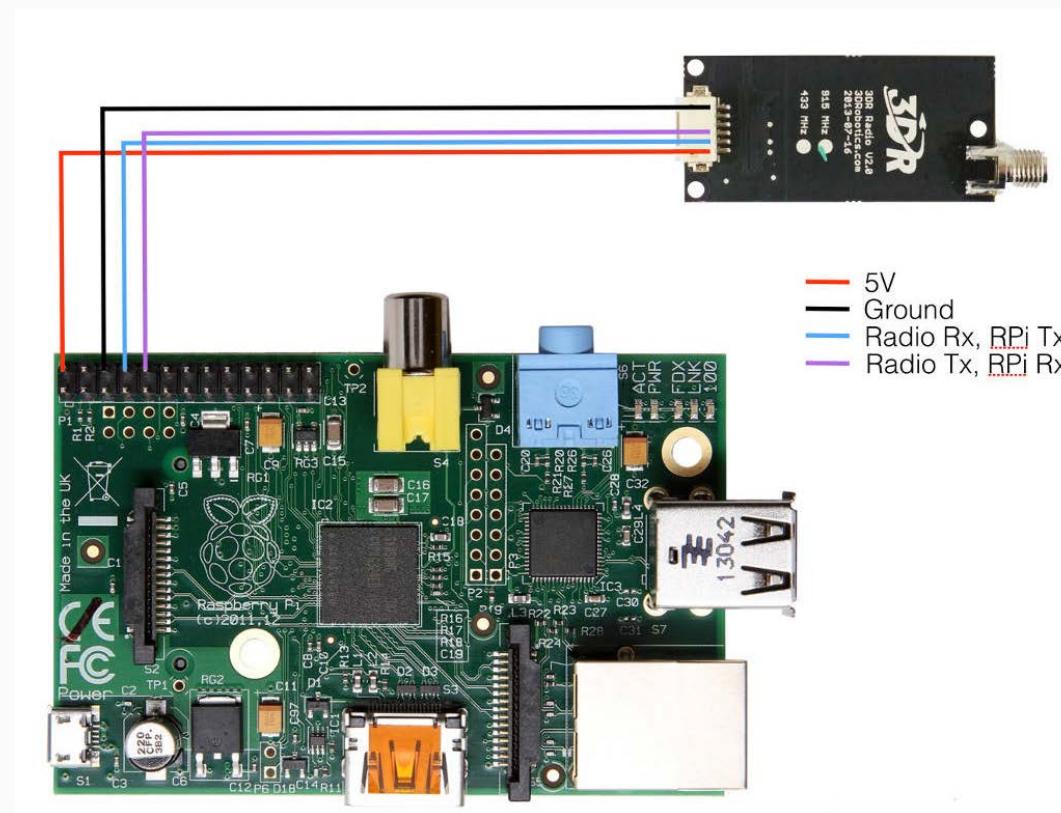
`/dev/ttyUSB0` (USB connection) rather than `/dev/ttyAMA0` (UART).

In order to communicate with a vehicle we will use the serial connection on the RPi. The serial port UART goes to pins 8 (TXD) and 10 (RXD) on the P1 header. We also need to provide power to the 3DR Radio by wiring pins 1 (+5V) and pin 6 (GND).

The 3DR Radio V2 pins 1 (+5V), 2 (RXD), 3 (TXD) and 6 (GND) are the corresponding pins to wire.

Remember that the TXD on the Raspberry Pi needs to be wired to the RXD on the 3DR Radio.

You can find more information on the pin out for the Raspberry Pi [HERE](#)



## Connecting Telemetry Radio to RaPi

Testing the MavProxy Connection

Now we are ready to test the communication. To do this follow the instructions on the section "[Testing the](#)

Connection" in the Raspberry Pi via Mavlink page.

Configuring MavProxy to always run and listen to incoming connections¶

The next step to get this working is to setup MavProxy to run automatically with the RPi boots up. To do this MavProxy has a daemon mode that works similarly to the above configuration for the DHCP server. In order to set it up we will use a script and modify it to work with the RPi.

1. Download the [mavgateway file](#) (from Github) and then copy it over to the RPi.
2. We now have to edit the file and change some aspects of it to make it compatible. Lets edit the file:

```
nano mavgateway
```

3. To setup the correct parameters to start MavProxy find the following line:

```
DAEMON_ARGS="--master=/dev/ttyMFD1,115200 --out=udpin:0.0.0.0:14550 --daemon"
```

- If you're using a serial connection change it to this:

```
DAEMON_ARGS="--master=/dev/ttyAMA0,57600 --out=udpin:0.0.0.0:14550 --daemon"
```

- If you're using a USB connection change it to this:

```
DAEMON_ARGS="--master=/dev/ttyUSB0,57600 --out=udpin:0.0.0.0:14550 --daemon"
```

4. Next we need to change the user that starts MavProxy, find this line:

```
start-stop-daemon --start --background --make-pidfile --chuid edison --chdir /tmp --quiet --pidfile
$PIDFILE --exec $DAEMON -- \
```

And change it to:

```
start-stop-daemon --start --background --make-pidfile --chuid pi --chdir /tmp --quiet --pidfile
$PIDFILE --exec $DAEMON -- \
```

Save the file by typing in **Control-X** then **Y** then **return**

5. Now we need to move the file and set the correct permissions. Type the following commands:

```
sudo mv mavgateway /etc/init.d/mavgateway
sudo chown root:root mavgateway
sudo chmod 755 mavgateway
```

6. We are now ready to setup the daemon to run every time the RPi boots type this command:

```
update-rc.d mavgateway defaults
```

7. Reboot the RPi:

```
sudo reboot
```

Connecting to the MavGateway¶

MavProxy¶

Using MavProxy (replace **xxx.xxx.xxx.xxx** with the IP Address of the RPi):

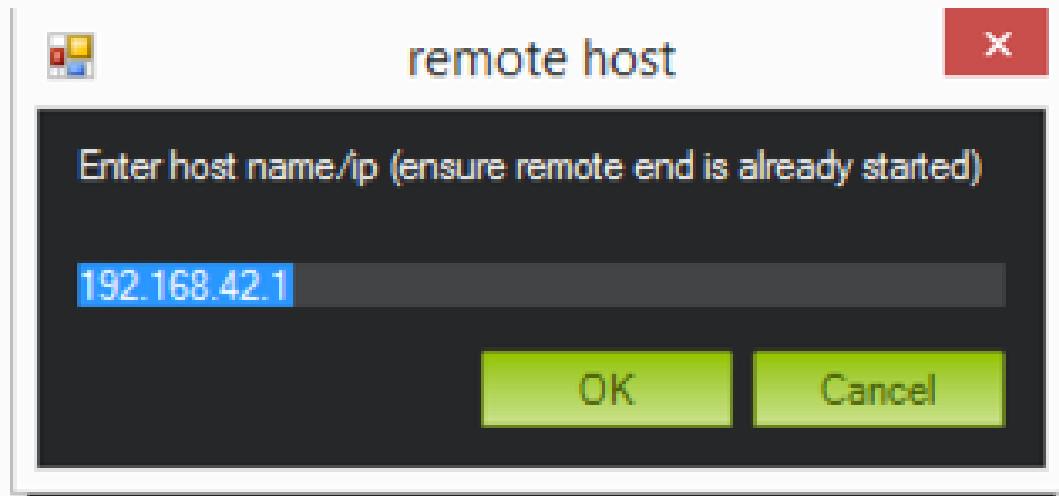
```
mavproxy.py -master=udpout:xxx.xxx.xxx.xxx:14550
```

### Using Mission Planner

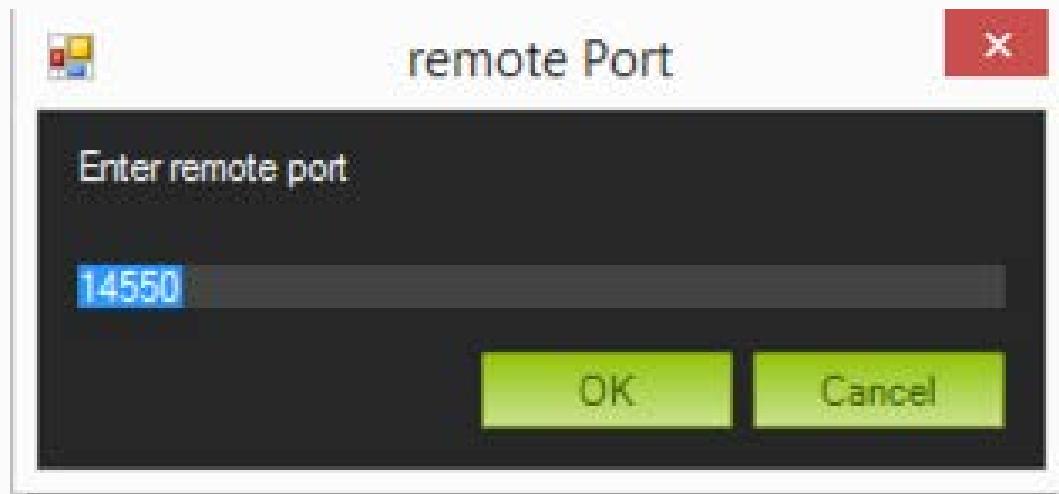
Using the latest beta version (1.3.17.1 build 1.1.5478.13250) you can now initiate a UDP connection. To do this select **UDPCI** from the connection type menu:



After that click connect, you will be presented with a screen to enter the IP Address of the RPi:



And enter the UDP Port on the next screen:



### Using APM Planner – TBD

Using a Tablet – TBD

SD Card Image file

If you don't want to configure your own RPi, you can download this image file and restore it to an SD Card to use it on your own RPi from this [link](#).

To restore **this image** to an SD Card you will need an SD Card with at least 16GB of space, just follow the same steps described in the section "[Getting the Raspberry Pi up and running with Raspbian](#)", but use this image instead of the OS image.

#### References¶

<https://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/install-software>

<http://sirlagz.net/2013/02/10/how-to-use-the-raspberry-pi-as-a-wireless-access-pointrouter-part-3b/>

## Communicating with ODroid via MAVLink

### Overview

This page introduces how to connect and configure an **ODROID U3**, **ODROID-C1+** or **ORDOID-XU4** so that it is able to communicate with a Pixhawk flight controller using the MAVLink protocol over a serial connection. The capabilities are very similar to the [Raspberry Pi](#) except that the ODroid has a much faster CPU (performance is roughly 10x faster than the RPi).



### Recommended parts

These parts are required:

- ODROID board ([ODROID-C1+](#), [ORDOID-XU4](#) or [ODROID U3](#))
- [5V Power supply](#)
- [DC Plug and Cable Assembly](#)
- [8GB MicroSD Card](#) OR [8GB eMMC Module](#) preloaded with Linux (the EMMC module is about 4x faster which may be useful for image processing)
- [FTDI Cable](#) or USB to Serial interface like [this one](#) from ebay

These optional items may also be useful:

- [Logitech HD Pro Webcam C920](#) is the recommended camera for image processing applications because it natively encodes images using the h264 format.

### Note

The usable resolution of this webcam is limited to 640 x 480 (or perhaps 1280 x 720) because no official Linux driver has been written for this camera yet.

- [Protective Case](#)
- [Micro HDMI cable](#) in case you wish to connect the ODroid to a computer monitor
- [Wifi module](#) if you wish to connect to the ODroid from your desktop computer using WiFi
- [Serial Debug cable](#) *if you wish to connect to the ODroid console with a direct connection to your desktop computer*
- Cat6 cable for connection while on the ground

## Initial ODROID Setup

It is easier to manage the odroid after it has a connection to your network. We've used the following steps to complete the initial setup of the board:

### Discovering IP Address

#### Using Keyboard/Mouse/Monitor

- Plug the preloaded eMMC or MicroSD card into the board
- (Temporarily) attach a mouse and keyboard to the USB ports
- Plug power into the board (at this point the boot LED should light up blue)
- Attach micro-HDMI cable to the board (this must be done *after* the board starts to boot, or the monitor may backpower the board / keeping it from booting)
- A text menu will appear
  - Choose advanced settings and enable the SSH server
  - Choose the option to expand the root filesystem to fill the card
  - Select save and reboot
- The board will reboot into a fairly standard looking Ubuntu
- Go to the network setup icon in the upper right and join your wifi network
- Type "ifconfig" to see the IP address your board was assigned

#### Using command line

You can discover the ip address of devices on your network using a utility called nmap

```
$ sudo nmap -sn 10.0.1.0/24
```

This will return a list of discovered devices on your network for and you will see an entry like

*\*(this example is RPi - will update to ODROID)*

```
Nmap scan report for 10.0.1.17
Host is up (0.00056s latency).
MAC Address: B8:27:EB:0A:26:9D (Raspberry Pi Foundation)
```

## Common Setup with known IP Address

- From some other computer type "ssh -Y [odroid@IPADDRESS](#)" with password odroid. Confirm you can login and get a ssh shell
- At an odroid shell type "sudo adduser odroid dialout" - this gives the odroid account access to the serial (pixhawk) port.
- At an odroid shell type "sudo adduser odroid plugdev" - this gives the odroid account access to the USB camera.
- Now you can unplug the keyboard/mouse/hdmi cables and just use ssh if you want. As long as you

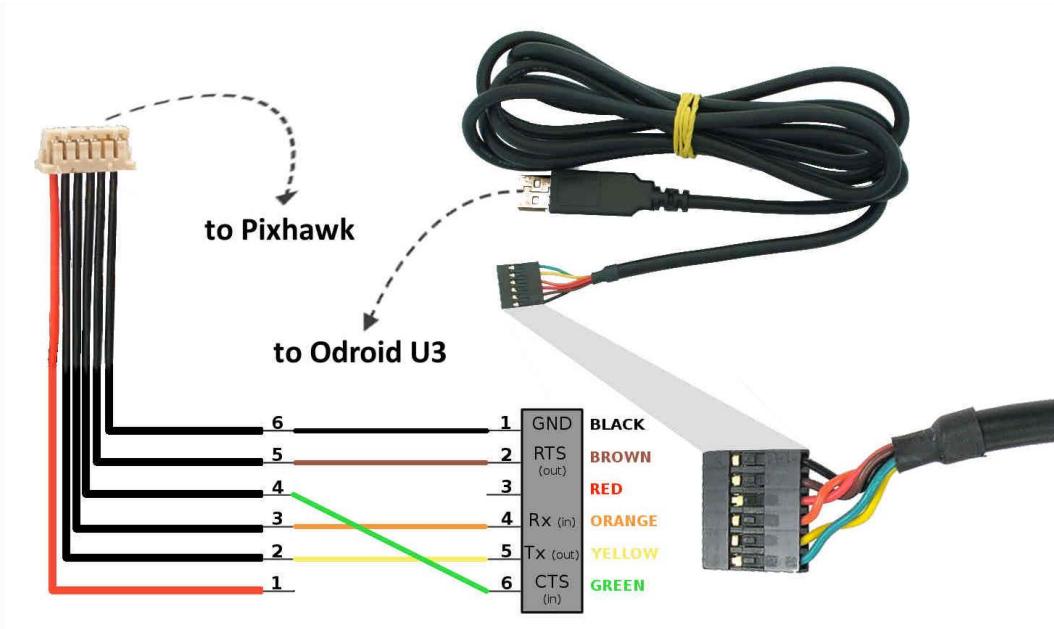
use the -Y option with SSH you can even run graphical applications on the odroid and they will appear on your desktop computer (assuming you are running something that understands X-Windows)

## Connecting the Pixhawk and ODroid



If using an FTDI cable, connect as shown in the picture below, DO NOT connect the power wire.



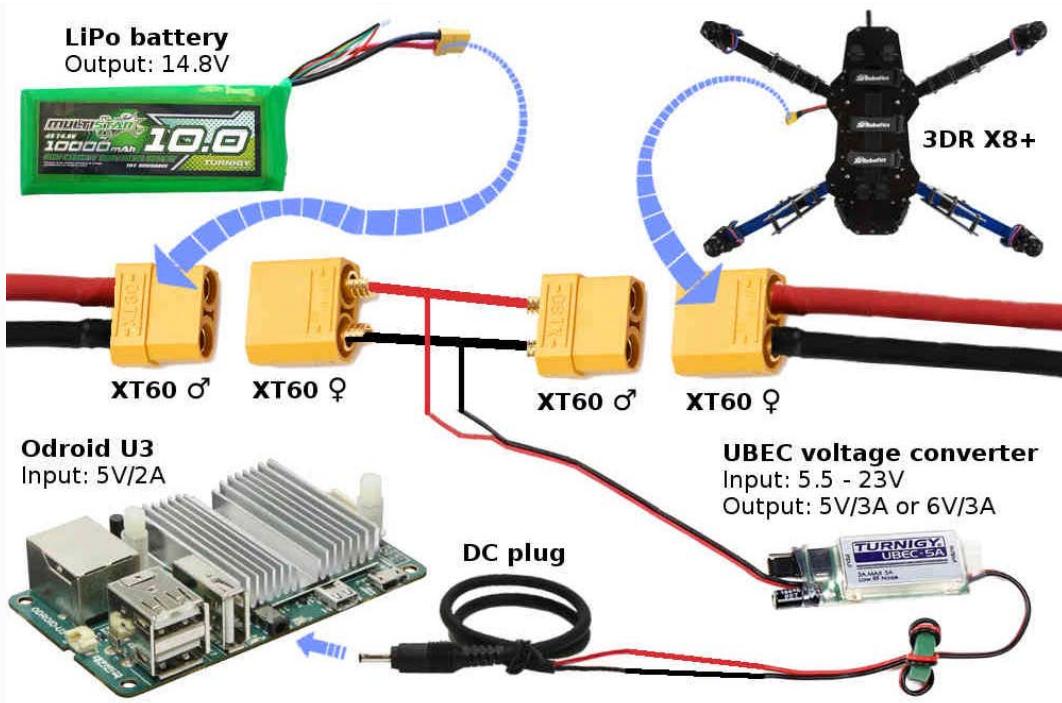


### ***Odroid FTDI/USB cable (detailview)***

If using a USB to Serial interface, connect as shown below.



Power connections are shown below:



## ***Odroid Power Connections***

### **Connecting to ODroid with an SSH/Telnet client**

To-Do: confirm the RPi instructions for all sections below also work for the ODroid.

Please refer to the instructions for the Raspberry Pi.

### **Installing the required packages**

Please refer to the instructions for the Raspberry Pi but skip the “Disable the login prompt” section.

### **Testing the connection**

Before attempting this, you will need to connect to your Pixhawk via mission planner with the USB cable.

Set SERIAL2\_BAUD (telemetry 2 baud rate) to 1500 (1.5Mbit)



To test the ODroid and Pixhawk are able to communicate with each other first ensure both are powered, then in a console on the Odroid type:

```
mavproxy.py --master=/dev/ttyUSB0 --baudrate 1500000 --aircraft MyCopter
```

once mavproxy has started you should be able to type in the following commands to arm and then disarm the copter.

```
arm throttle
```

disarm

```

root@odroid: ~
login as: odroid
odroid@192.168.1.13's password:
Welcome to Ubuntu 13.10 (GNU/Linux 3.8.13.16 armv7l)

 * Documentation:  https://help.ubuntu.com/

Last login: Thu Apr 17 20:03:59 2014 from randy-xps.fletsphone
odroid@odroid:~$ sudo -s
[sudo] password for odroid:
root@odroid:~# mavproxy.py --master=/dev/ttyUSB0 --baudrate 57600 --aircraft MyCopter
MyCopter/logs/2014-04-17/flight2
Logging to MyCopter/logs/2014-04-17/flight2/flight.tlog
no script MyCopter/mavinit.scr
Loaded module log
Loaded module rally
Loaded module fence
MAV> UNKNOWN> Mode UNKNOWN
waypoint 65535
APM: PX4v2 001A002E 34324709 31323533
Received 367 parameters
Saved 367 parameters to MyCopter/logs/2014-04-17/flight2/mav.parm
STABILIZE> Mode STABILIZE
MAV> arm throttle
STABILIZE> APM: Calibrating barometer
APM: barometer calibration complete
Got MAVLink msg: COMMAND_ACK {command : 400, result : 0}
ARMED
disarm
STABILIZE> Got MAVLink msg: COMMAND_ACK {command : 400, result : 0}
DISARMED

```

## Configure mavproxy to always run

This is accomplished by adding the following to the end of your "/home/odroid/.bashrc" file. The Pixhawk is connected via "ttyUSB0". The last section of the file should look like below. **Change the IP address to that of your ground station** (this also starts the balloon finder, comment that line out if you don't want it to run)

```

export PATH=$PATH:$HOME/GitHub/ardupilot-balloon-finder
export PATH=$PATH:$HOME/GitHub/ardupilot-balloon-finder/scripts
export PYTHONPATH=$PYTHONPATH:$HOME/GitHub/ardupilot-balloon-finder/scripts
if screen -list | grep -q "MavProxy"; then
    echo "MavProxy is already running"
else
    cd /home/odroid
    screen -S MavProxy -d -m -s /bin/bash mavproxy.py --master=/dev/ttyUSB0 --baudrate 1500000 --out
    192.168.1.11:14550 --aircraft MyCopter
    python /home/odroid/GitHub/ardupilot-balloon-finder/scripts/colour_finder_web.py
    echo "started MavProxy type screen -x to view"
fi

```

## Connecting with the Mission Planner

Please refer to the instructions for the Raspberry Pi.

## Example Projects

### Red Balloon Finder

The Red Balloon Finder project was written to enable an Odroid to control an Copter based quadcopter to hunt down and pop 1m red balloon for the [Sparkfun 2014 AVC](#). The python code that runs on the odroid can be found [here](#). The slightly modified Copter-3.2 code can be found [here](#). Installation

instructions are below.

Set the following parameters on the Pixhawk:

SERIAL2\_BAUD to 921 (921600 bits per sec)

SR2\_EXTRA1 to 20 (20hz update rate for attitude from Pixhawk to Odroid)

Create a home/odroid/GitHub directory where we will install the required source code:

```
cd /home/odroid/
mkdir GitHub
cd GitHub
```

Type the following commands to download the ardupilot-balloon-finder source code:

```
git clone https://github.com/rmackay9/ardupilot-balloon-finder.git
```

Add the ardupilot-balloon-finder to the PATH and PYTHONPATH by doing the following:

```
cd /home/odroid/
nano .bashrc      #Note that "nano" can be replaced by your favourite editor such as "vi"
```

scroll to the bottom of the file and add the following as the very last items

```
export PATH=$PATH:$HOME/GitHub/ardupilot-balloon-finder
export PATH=$PATH:$HOME/GitHub/ardupilot-balloon-finder/scripts
export PYTHONPATH=$PYTHONPATH:$HOME/GitHub/ardupilot-balloon-finder/scripts
```

## Drone API

To install the latest Drone API , run the following:

```
sudo pip install --upgrade droneapi
```

next setup mavproxy to start the DroneAPI each time it is started:

```
nano /home/odroid/MyCopter/.mavinit.scr
```

then add the following to the file (correct the path of the balloon\_strategy.py to your path)

```
module load droneapi.module.api
api start /home/odroid/GitHub/ardupilot-balloon-finder/scripts/balloon_strategy.py
```

## Cherrypy

Cherrypy is a simply python webserver used by ardupilot-balloon-finder to show what the camera is currently seeing on a web page. Install cherrypy by entering the following in an Odroid terminal:

```
sudo pip install cherrypy
```

Can't get it to work? Try posting your question in the [APM Forum's APM Code section](#).

ODroid Wifi Access Point for sharing files via Samba

## Overview

This page will show you how to setup an Odroid with a Wifi AccessPoint so that the Odroid's hard drive can be accessed and modified from another computer. This is primarily aimed at allowing access to images, videos and log files on the Odroid. The procedure makes use of [hostapd](#), [Samba](#) and [dhcp](#).

To accomplish this you will need:

- Odroid U3 running Ubuntu 14.04 as described on [this page](#).
- A Wifi dongle capable of "master" mode (aka "AP" or access point). These exact instructions were performed with an \$8 [Buffalo Airstation 11n 11g/b USB2 wireless LAN terminal purchased in Japan](#).

All the instructions below can be implemented by connecting a keyboard, mouse and screen to the Odroid or using ssh (via Putty).

### Install all the required packages

Becomes super user:

```
sudo -s
```

Install hostapd, samba and all the other required packages:

```
apt-get install hostapd samba samba-common python-glade2 system-config-samba isc-dhcp-server
```

### Setting up the Access Point

After first ensuring the wifi dongle is plugged into the Odroid,

Find the name of your wifi dongle by typing `ifconfig` (normally it will be "`wlan0`" or "`wlan2`")

```

login as: odroid
odroid@192.168.1.13's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.8.13.23 armv7l)

 * Documentation:  https://help.ubuntu.com/

365 packages can be updated.
145 updates are security updates.

Last login: Sun Jan 18 14:29:11 2015 from 192.168.100.100
odroid@odroid:~$ ifconfig
eth0      Link encap:Ethernet HWaddr c2:22:09:f2:5f:e8
          inet addr:192.168.1.13 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: 2408:212:6a80:5100:c022:9ff:fef2:5fe8/64 Scope:Global
          inet6 addr: 2408:212:6a80:5100:90a4:c181:18d2:1341/64 Scope:Global
          inet6 addr: fe80::c022:9ff:fef2:5fe8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:8971 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6622 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8193535 (8.1 MB) TX bytes:3590137 (3.5 MB)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:40349 errors:0 dropped:0 overruns:0 frame:0
          TX packets:40349 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2420956 (2.4 MB) TX bytes:2420956 (2.4 MB)

wlan2    Link encap:Ethernet HWaddr cc:e1:d5:17:8d:da
          inet addr:192.168.100.1 Bcast:192.168.100.255 Mask:255.255.255.0
          inet6 addr: fe80::ceec1:d5ff:fe17:8dda/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:352 errors:0 dropped:0 overruns:0 frame:0
          TX packets:492 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:54295 (54.2 KB) TX bytes:87593 (87.5 KB)

odroid@odroid:~$ 

```

Check the wifi dongle supports “AP” mode:

```
iw list
```

A huge list of information will be displayed, you are looking for a section called “Supported interface modes:” followed by “\* AP”. If you cannot find this then these instructions will not work and you should try with another dongle.

```

Supported interface modes:
* IBSS
* managed
* AP

```

Use your favourite editor (nano or vi perhaps) to edit the `/etc/network/interfaces` file and add an entry for your wifi dongle. Note you should replace “wlan2” if your dongle has a different name:

```
auto wlan2
```

```
iface wlan2 inet static
```

```
address 192.168.100.1
```

netmask 255.255.255.0

Edit the `/etc/hostapd/hostapd.conf` file and ensure it looks like below

```
vi /etc/hostapd/hostapd.conf

interface=wlan2      <-- change "wlan2" to name of wifi dongle found above
driver=nl80211
ssid=MyDrone
hw_mode=g
channel=11
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=MyDrone12    <-- password for clients to access the wifi access point
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Try starting hostapd:

```
hostapd /etc/hostapd/hostapd.conf
```

If this fails try replacing the `/usr/sbin/hostapd` and `hostapd_cli` files with the version found in the [downloads area](#).

You will also need to install the `libnl-dev` package:

```
apt-get install libnl-dev
```

To make the hostapd service run whenever the odroid is started edit the `/etc/init.d/hostapd` file and ensure the line below appears:

```
DAEMON_CONF=/etc/hostapd/hostapd.conf
```

Setting up Samba¶

Create directories you wish to share:

```
mkdir -p /mydrone
mkdir -p /mydrone/images
```

Edit the samba config file (if not present create a new file):

```
vi /etc/samba/smb.conf
```

```
[global]
workgroup = MyDroneGroup
server string = Drone Server
netbios name = mydrone
security = user
map to guest = bad user
dns proxy = no
===== Share Definitions =====
[images]
path = /mydrone/images
```

```
browsable = yes
writable = yes
guest ok = yes
read only = no
```

The this config will create a “images” share with no restriction on clients adding or deleting files.

### Setting up DHCP¶

Edit the `/etc/dhcp/dhcpd.conf` file and ensure it has all the lines listed below added/uncommented:

```
vi /etc/dhcp/dhcpd.conf
```

```
# option definitions common to all supported networks...
option domain-name "mydrone.local";
option domain-name-servers dns.mydrone.local
default-lease-time 600;
max-lease-time 7200;
authoritative;
log-facility local7;
subnet 192.168.100.0 netmask 255.255.255.0 {
    range 192.168.100.100 192.168.100.200;      <-- clients will get IP addresses in this range
}
```

Add all possible client IP addresses to the `/etc/hosts` file:

```
vi /etc/hosts
```

```
127.0.0.1 localhost
192.168.100.1 odroid-pc odroid
192.168.100.100 client100
192.168.100.101 client101
192.168.100.102 client102
...
192.168.100.200 client200
```

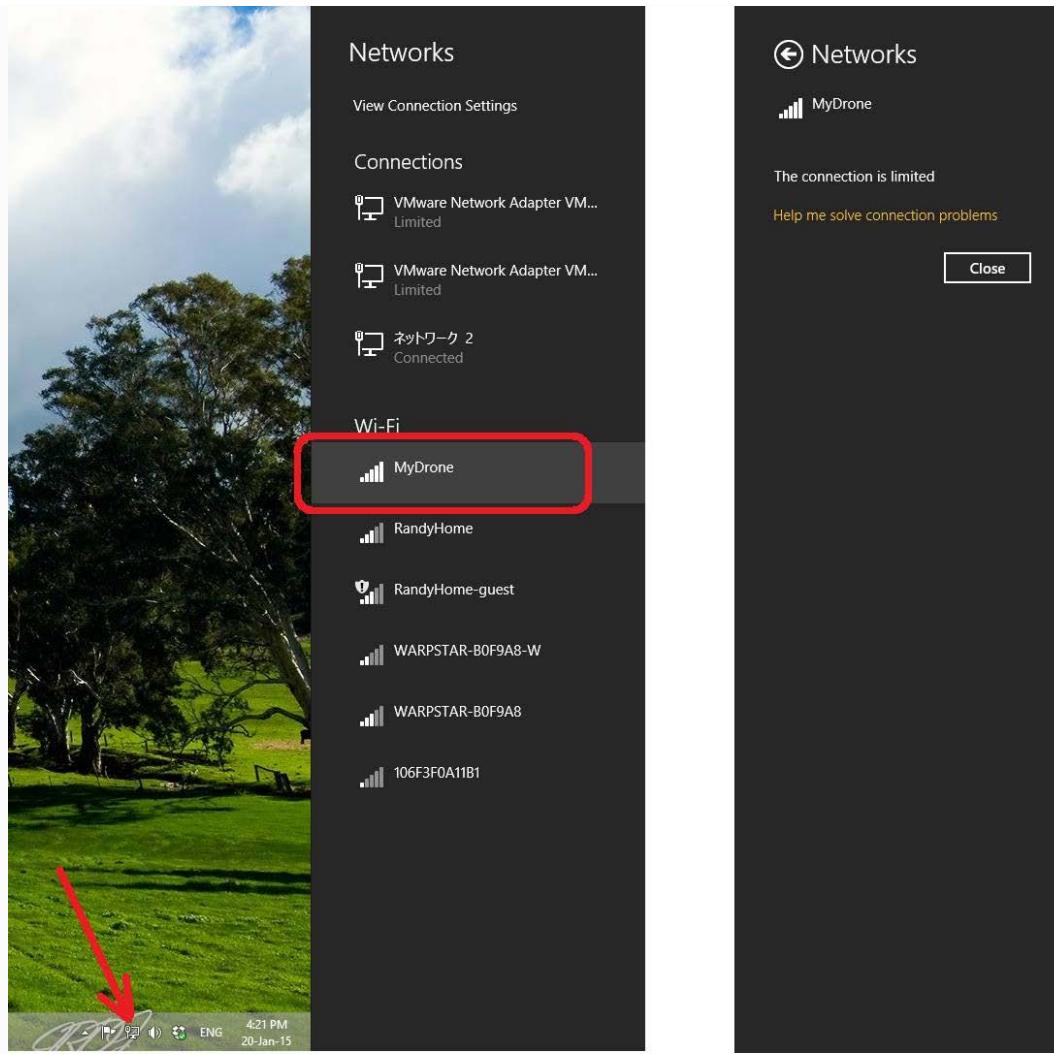
Adding all the client IP address can be accomplished more quickly with the following command:

```
(for i in $(seq 100 200); do echo 192.168.100.$i client$i; done) >> /etc/hosts
```

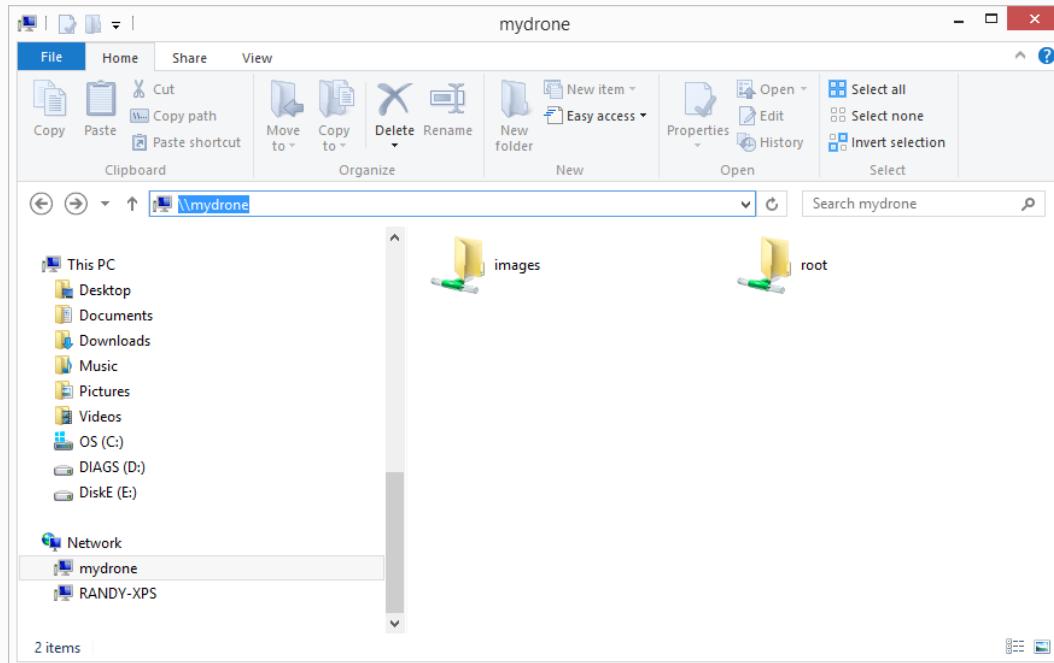
Restart the Odroid and with any luck an AP will be created and you will be able to connect as shown below.

### Connecting from a Windows PC¶

To connect from a Windows 8.1 machine click on the network icon near the clock and then click on “MyDrone”, “Connect” and after 30seconds or so it should connect with a message “The connection is limited” because the Odroid likely does not have access to the internet (this is ok).



Open a File Explorer and type \\mydrone into the address bar and the contents of the Odroid images directory should appear.



## References

These pages were referenced during the creation of this document.

wireless.kernel.org's hostapd documentation.

## Intel Edison as a Companion Computer

This page explains how to setup and use the [Intel Edison](#) as a companion computer primarily for use with the [Pixhawk2](#).



The Intel® Edison can provide features including:

- Wifi telemetry to the autopilot
- Easy scripting/vehicle control via DroneKit
- Faster download of log files (coming soon)

## Where to buy

The Edison can be [purchased from Sparkfun here](#).

The Pixhawk2 with carrier board can be purchased from one of these [retailers](#). Be sure to purchase a Pixhawk2 with the Intel Edison compatible carrier board.

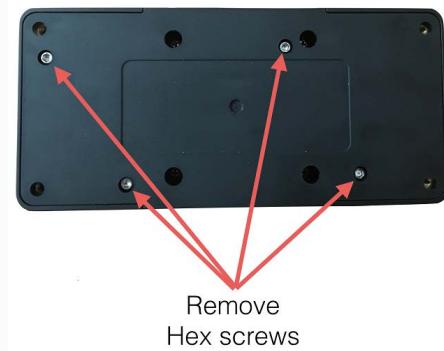
## Installing the Edison in the Pixhawk2

To install the Edison into the Pixhawk 2 Carrier board:

- Remove the four side screws from the Pixhawk2



- Remove the four hex screws on the bottom of the carrier board and lift the bottom cover to reveal the socket for the edison.



- Remove the philips screws that will later hold the Edison in place. Place the Intel Edison into its socket and use the screws to hold it in place. It should now look like the top-most image on this page (see above)
- Replace the bottom cover in the reverse order to the instructions above

## Setting up the Edison

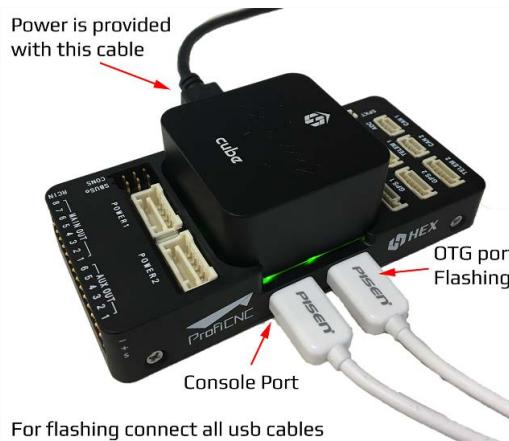
### Note

Some information here was borrowed from [the Intel Developer Zone](#).

The easiest way to get started is to flash the Edison with image recommended by the ArduPilot team:

- Download the [latest image from firmware.ardupilot.org](#)
- Extract/Unzip the image (a “toFlash” directory should appear):
  - Windows users can use [7-zip](#)
  - Ubuntu users can right-mouse-button-click and select “Extract Here” or type
 

```
tar -xvf intel_edison_image_latest.tar.gz
```
- Install dfu:
  - On Windows:
    - Download and extract [dfu-util-0.9.win64.zip](#) from [dfu-util.sourceforge.net/releases](#)
    - Copy the [dfu-util.exe](#) and [libusb-1.0.dll](#) files into the [toFlash](#) directory created when extracting the image (see above)
    - Download and install the latest [drivers from Intel](#).
  - On Ubuntu install with [`sudo apt-get install dfu-util`](#)
  - On OS X:
    - Follow the instructions on the [Homebrew web page](#).
    - Install dfu-util [`brew install dfu-util`](#)
- Connect your PC to the Pixhawk2 using the USB cables as shown below. This provides power and enables flashing the image



- The ports on the carrier board connect to the Edison, one is the serial console port and the other is the OTG port. See the image below to identify each one:



- Flash the image:
  - On Windows double click on the `flashall.bat` script found in the `toFlash` directory or Open a command prompt window navigate to the `toFlash` folder and run `flashall.bat` to see the output
  - On Ubuntu and OS X cd into the `toFlash` directory and enter, `./flashall.sh`
  - During the flashing process, the script will ask you to un plug the Edison. For this you must cut power to the Cube by removing only the USB cable connected to the cube itself.
- After flashing is done, wait 1 to 2 min before cutting power to the Edison
- This video shows how to do this process on a OS X machine, but the process should be very similar in Linux and Windows.

After flashing has completed the root file system must be expanded manually from 1.5GB to 2.2GB:

- Connect two USB cables from your PC to the Pixhawk2, one to the cube, the other one to the console port.
- Open a serial connection to the Edison (which uses the 2nd USB connection) at 115200 baud with username and password "edison"
  - On windows you may use Putty
  - On Linux/Ubuntu or OSX you can use screen, `screen /dev/tty.usbserial-Axxxxxxx 115200` ("xxxxxxxx" value is specific to each board)
- use the `.local/bin/post-flash.sh` script to expand the file system:

```
edison@edison ~ $ .local/bin/post-flash.sh
Running post install chores
[sudo] password for edison:
resize2fs 1.42.12 (29-Aug-2014)
Filesystem at /dev/mmcblk0p8 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcblk0p8 is now 589824 (4k) blocks long.
```

Finally edit these files with your Wi-Fi network credentials:

```
/etc/network/interfaces.home
/etc/network/interfaces.work
```

Then you can log into the Edison and type `homenet.sh` or `worknet.sh` to switch between network configurations

## Troubleshooting bricked Edison

In some cases the Edison may stop responding to the flashing script. If this happens you might want to try to recover the Edison by doing the following. It is important to note this will only work under Linux Ubuntu 14.04

Download the latest version of xFSTK onto your Ubuntu 14.04 32-bit system from [here](#). and extract.

1. Unzip the downloaded file with

```
tar xvfz xfstk-dldr-linux-source-1.7.2.tar.gz
```

2. Navigate to the source folder

```
cd xfstk-build/linux-source-package
```

3. Install the required packages

```
sudo apt-get install g++ qtcreator build-essential devscripts libxml2-dev alien doxygen graphviz libusb-dev
sudo apt-get install libqt4-dev qt4-qmake
sudo apt-get install libusb-1.0-0-dev
```

4. Create the following Symlink

```
ln -s /usr/lib/x86_64-linux-gnu/libusb-1.0.a /usr/lib/libusb.a
```

5. Configure the build parameters

```
export DISTRIBUTION_NAME=ubuntu14.04
export BUILD_VERSION=0.0.0
```

6. Build the xFSTK tools

```
make --version -j 6
```

7. Run cmake

```
mkdir build
cd build
cmake ..
```

8. Build the package

```
make package
```

## 9. Install the package you just built

```
dpkg -i [built package]
```

## 10. May need to install:

```
sudo apt-get install libboost-program-options1.55.0
sudo apt-get install dfu-util
```

now you should be able to run `./flashall.sh --recovery` to recover the Edison.

## Archived Instructions

The following instructions were written before the standard image was created and are not useful for most users.

[Download](#) the Edison SDK appropriate for your platform and install it:

```
kevinh@kevin-think:~/tmp$ sudo ./poky-edison-eglibc-x86_64-edison-image-core2-32-toolchain-1.6.sh
[sudo] password for kevinh:
Enter target directory for SDK (default: /opt/poky-edison/1.6):
Extracting SDK...
```

The default Edison load is missing a number of useful features, so we recommend using the [ubilinux](#) port of Debian.

1. Download the latest ubilinux [build](#).
2. Install per their [instructions](#)

```
kevinh@kevin-think:~/development/drone/edison/new/ubilinux-edison-ww44.5$ sudo ./flashall.sh
[sudo] password for kevinh:
Using U-Boot target: edison-blank
Now waiting for dfu device 8087:0a99
Flashing IFWI
... lots of stuff ...
Your board needs to reboot twice to complete the flashing procedure, please do not unplug it for 2 minutes.
```

3. Have the Edison join your wifi network: On the serial console of the edison login with username **root**, password **edison**. "vi /etc/network/interfaces". And adjust settings for your local wifi network name and password (and uncomment the line that says "auto wlan0"). Then save the file and run "ifup wlan0".

You will see:

```
...
DHCPREQUEST on wlan0 to 255.255.255.255 port 67
DHCPoffer from 192.168.1.1
DHCPACK from 192.168.1.1
bound to 192.168.1.37 -- renewal in 40603 seconds.
```

Your Edison will now be on your wifi network at the indicated IP address. You can disconnect your serial session and connect to Edison via ssh (much easier/faster):

```
ssh root@IP-ADDRESS-SEEN-ABOVE
The authenticity of host '192.168.1.37 (192.168.1.37)' can't be established.
ECDSA key fingerprint is af:f2:ae:e4:7f:0d:b4:42:3d:c6:db:ac:e7:c7:66:bb.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.37' (ECDSA) to the list of known hosts.
root@192.168.1.37's password:
Linux ubilinux 3.10.17-poky-edison-ww42+ #4 SMP PREEMPT Wed Oct 29 12:41:25 GMT 2014 i686
...
```

Now install Dronekit, Opencv and Mavproxy.

```
root@ubilinux:~#
root@ubilinux:~# apt-get update
... lots of messages from apt ...
root@ubilinux:~# apt-get dist-upgrade
... lots of messages from apt ...
root@ubilinux:~# apt-get install git sudo python-pip python-numpy python-opencv python-serial python-pyparsing
... lots of messages from apt ...
root@ubilinux:~# pip install droneapi
... lots of messages from PIP ...
Successfully installed droneapi pymavlink MAVProxy protobuf
Cleaning up...
```

And add the 'edison' user account to the various groups it should be a member of so it can do dangerous things.

```
root@ubilinux:~# usermod -a -G sudo,plugdev,dialout edison
```

Configure mavproxy to always load the DroneAPI module:

```
echo "module load droneapi.module.api" > ~/.mavinit.scr
```

Install the DroneAPI example code

```
kevinh@kevin-think:~/development/drone/edison/new/ubilinux-edison-ww44.5$ ssh edison@192.168.1.37
edison@ubilinux:~$ git clone https://github.com/dronekit/dronekit-python.git
Cloning into 'droneapi-python'...
remote: Counting objects: 460, done.
remote: Total 460 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (460/460), 246.43 KiB | 182 KiB/s, done.
Resolving deltas: 100% (213/213), done.
edison@ubilinux:~$ cd droneapi-python/example
edison@ubilinux:~/droneapi-python/example$ ls
client_sketch.py follow_me.py run-fake-gps.sh fake-gps-data.log microgcs.py small_demo.py
```

Run a basic test of your coprocessor/DroneKit connection (ttyMFD1 is the serial port connecting the Edison to the Pixhawk):

```
edison@ubilinux:~/dronekit-python/example$ mavproxy.py --master=/dev/ttyMFD1,57600 --rtscts
Logging to mav.tlog
libdc1394 error: Failed to initialize libdc1394
Failed to load module: No module named terrain
Running script /home/edison/.mavinit.scr
-> module load droneapi.module.api
DroneAPI loaded
Loaded module droneapi.module.api
MAV> Flight battery warning
AUTO> Mode AUTO
APM: Plane V3.2.0 (a9defa35)
APM: PX4v2 002E001B 3433470D 32323630
T|D0 .Received 486 parameters
```

```

AUTO> api start small_demo.py (If you see the messages below your vehicle is now happily talking to
mavproxy/dronekit)
AUTO> mode is AUTO
Mode: VehicleMode:AUTO
Location: Location:lat=0.0,lon=0.0,alt=1.38999998569,is_relative=False
Attitude: Attitude:pitch=0.00390338362195,yaw=-1.69979262352,roll=-3.12372088432
Velocity: [0.0, 0.0, 0.0]
GPS: GPSInfo:fix=0,num_sat=0
Armed: False
groundspeed: 0.0
airspeed: 14.2826738358
Requesting 0 waypoints t=Fri Nov 28 19:42:14 2014 now=Fri Nov 28 19:42:14 2014
Home WP: MISSION_ITEM {target_system : 255, target_component : 0, seq : 0, frame : 0, command : 16, current
: 0, autocontinue : 1, param1 : 0.0, param2 : 0.0, param3 : 0.0, param4 : 0.0, x : 0.0, y : 0.0, z : 0.0}
Current dest: 0
Disarming...
Arming...
Overriding a RC channel
Current overrides are: {'1': 900, '4': 1000}
RC readback: {'1': 0, '3': 0, '2': 0, '5': 0, '4': 0, '7': 0, '6': 0, '8': 0}
 Cancelling override
mode is AUTO
APIThread-0 exiting...
APM: command received:
Got MAVLink msg: COMMAND_ACK {command : 400, result : 4}
APM: Throttle armed!
Got MAVLink msg: COMMAND_ACK {command : 400, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}

AUTO> edison@ubilinux:~/dronekit-python/example$
```

Done! Now you can run your own custom DroneKit code on the Edison (see the tutorial or documentation for more information).

## **Wifi mavlink bridging**

The Edison can forward mavlink to other computers on your wifi lan (and you can either run a traditional GCS on those computers or run DroneKit scripts on your PC).

To turn on this feature you need to leave mavproxy running indefinitely, listening for incoming UDP packets from clients.

update-rc.d mavgateway defaults

mavproxy.py –master=/dev/ttyMFD1,57600 –rtscts –out=udpin:0.0.0.0:14550

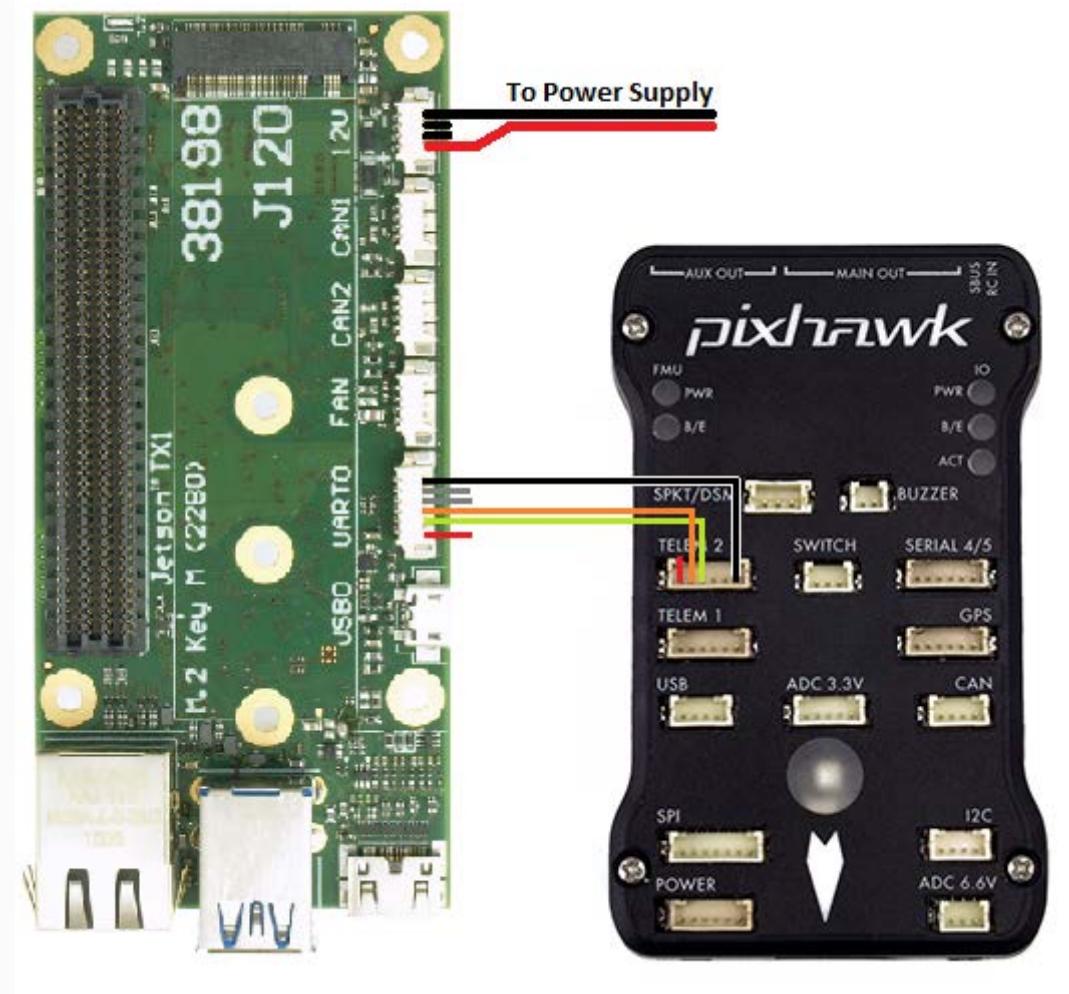
On your desktop computer:

mavproxy.py –master=udpout:192.168.1.37:14550

## **NVidia TX1 as a Companion Computer**

This page explains how to connect and configure an [NVidia TX1](#) using [AuVidea.eu's J120 carrier board](#) so that it is able to communicate with a Pixhawk flight controller using the MAVLink protocol over a serial connection.

### **Connecting the Pixhawk and TX1**



Connect the Pixhawk's TELEM2 port to the J120's UART0 port's Ground, TX and RX pins as shown in the image above.

The Pixhawk and TX1 should be powered separately (the J120/TX1 through it's 12V power input port, the Pixhawk through it's POWER port). They should be powered on at about the same time or the TX1 powered on first to avoid the Pixhawk interrupting the TX1's bootloader.

## Setup the Pixhawk

Connect to the Pixhawk with a ground station (i.e. Mission Planner) and set the following parameters:

- **TELEM\_DELAY** = 30. This delays the pixhawk from using the telemetry ports for 30 seconds. This is required to avoid interrupting the TX1's bootloader. Note this only works if the TX1 and Pixhawk are powered up at the same time (or the TX1 is powered up first).
- **SERIAL2\_BAUD** = 921. The Pixhawk and TX1 can communicate at 921600 baud.

## Setup the TX1

The easiest way to setup the TX1 is to flash one of the existing binaries from [firmware.ardupilot.org](http://firmware.ardupilot.org) (look for images starting with "tx1"). Note that the J120 boards do not allow flashing so instead an [Nvidia TX1 development board](#) must be used.

- mount the TX1 back on the Nvidia development board
- download and unzip the latest image starting with "tx1" from [firmware.ardupilot.org](http://firmware.ardupilot.org)
- official instructions on flashing images can be found [here](#) but in short:

- install the TX1 on an NVidia TX1 development board
- install JetPack on an Ubuntu machine
- connect a USB cable from the Ubuntu machine to the TX1 development board
- power on the TX1 development board
- put the TX1 into bootloader mode (Hold and keep pressed the “Force-Recovery” button, press and release the “Reset” button, release the “Force-Recovery” button). You can check the TX1 is in bootloader mode by typing “lsusb” on the Ubuntu machine and look for “NVidia”.
- on the Ubuntu machine, from the ..../JetPack/TX1/Linux\_for\_Tegra\_tx1/bootloader directory run a command like below where “IMAGE.img” is replaced with the name of the image file downloaded above:

```
sudo ./tegraflash.py --bl cboot.bin --applet nvtboot_recovery.bin --chip 0x21 --cmd "write APP IMAGE.img"
```

Note: instructions on how the firmware.ardupilot.org image was created can be found [here](#).

## BeaglePilot Project

The BeaglePilot Project aims to create the first Linux-based autopilot for flying robots using the BeagleBone and the BeagleBone Black as the “hardware blueprint”.

For this purpose the project will focus on integrating ArduPilot (most popular autopilot) in the BeagleBone (Black). The main tasks performed will be userspace drivers development, the Robot Operative System Integration (ROS), web IDE exploration and security assessment.

## Getting ArduPilot in the BeagleBone Black

Refer to the instructions provided in the article [Building \(ardupilot\) for BeagleBone Black on Linux](#).

## Platforms

For now BeaglePilot is still in development phase being tested in the following devices:

- BeagleBone Black + PixHawk Fire Cape
- [Erie Robot](#)

## Communication

- IRC Freenode #beaglepilot ([logs](#))
- [BeaglePilot mailing list](#)
- [eLinux Wiki](#)
- [BeagleBoard project page](#)
- [ArduPilot Wiki](#)
- [GitHub repository](#)

## Turnkey Companion Computer Solutions

This article lists turnkey companion computer solutions that are advertised as working with ArduPilot. These are typically commercially produced and tuned variations of open source solutions like BeagleBoard, ODroid, Raspberry Pi etc.

Note

Please let us know if you discover a new companion computer, so we can add it to this list.

## 4Gmetry Companion Computer Kit

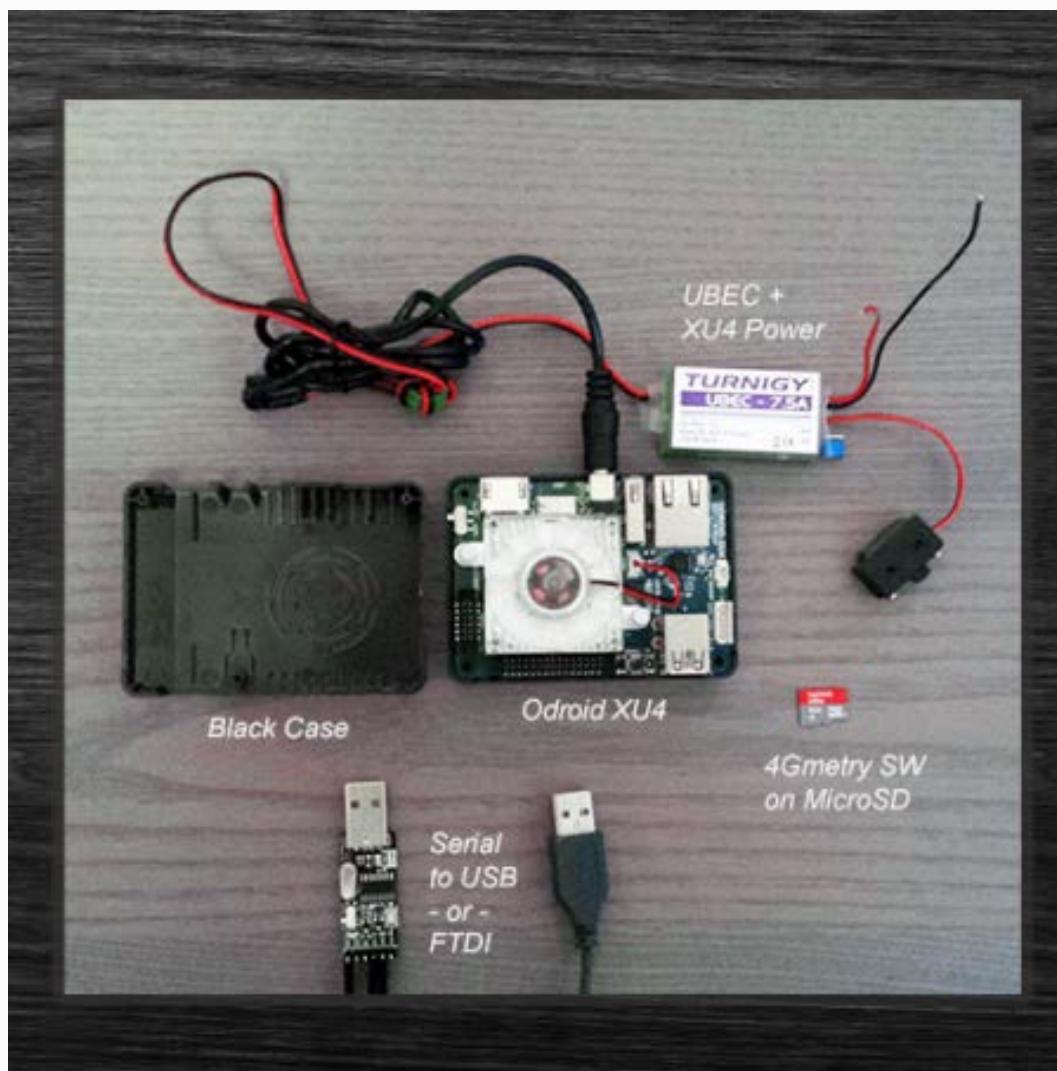
## Warning

The ArduPilot dev team advise that this product may not be as “turnkey” as indicated below. Community reports indicate that the Kit does not include the LTE module and may also not supply a powering mechanism.

[4Gmetry](#) (Volta Robots) is a plug&play companion computer kit, based on Odroid XU4 single board computer.

4Gmetry gets telemetry (MavLink) from the Autopilot and streams it over 4G internet to a remote control station. 4Gmetry is fully compatible with Volta OS to remotely manage fleets of robots via a simple high level API. 4Gmetry can be used for computer vision tasks and video/image streaming over internet.

4Gmetry comes ready to connect to your VPN, for safety/security purposes; this also allows you remote console access (e.g. SSH).



## 4GmetryII

*Image and text from Volta Website*

## EKF

Please follow the links below to learn more about ArduPilot's sophisticated attitude and position estimation system (the EKF).

## Extended Kalman Filter Navigation Overview and Tuning

This article describes the Extended Kalman Filter (EKF) algorithm used by Copter and Plane to estimate vehicle position, velocity and angular orientation based on rate gyroscopes, accelerometer, compass (magnetometer), GPS, airspeed and barometric pressure measurements. It includes both an overview of the algorithm and information about the available tuning parameters.

### Overview

The availability of faster processors such as Pixhawk and the PX4 have enabled more advanced mathematical algorithms to be implemented to estimate the orientation, velocity and position of the flight vehicle. An Extended Kalman Filter (EKF) algorithm has been developed that uses rate gyroscopes, accelerometer, compass, GPS, airspeed and barometric pressure measurements to estimate the position, velocity and angular orientation of the flight vehicle. This algorithm is implemented in the **AP\_NavEKF** library and is based on initial work documented here: <https://github.com/priseborough/InertialNav>

PX4 and Pixhawk users can use this algorithm instead of the legacy complementary filters by setting

`AHRS_EKF_USE = 1`.

#### Warning

**IMPORTANT:** Do not set `AHRS_EKF_USE = 1` unless you have performed an Accel and compass calibration.

Failure to do so may result in an erratic flight path due to bad sensor data.

The advantage of the EKF over the simpler complementary filter algorithms used by DCM and Copters Inertial Nav, is that by fusing all available measurements it is better able to reject measurements with significant errors so that the vehicle becomes less susceptible to faults that affect a single sensor.

Another feature of the EKF algorithm is that it is able to estimate offsets in the vehicles compas readings and also estimate the earth's magnetic field for both plane, copter and rover applications. This makes it less sensitive to compass calibration errors than current DCM and INAV algorithms.

It also enables measurements from optional sensors such as optical flow and laser range finders to be used to assist navigation.

### Theory

The EKF (Extended Kalman Filter) algorithm implemented, estimates a total of 22 states with the underlying equations derived using the following:

<https://github.com/priseborough/InertialNav/blob/master/derivations/GenerateEquations22states.m>

The following is a greatly simplified non-mathematical description of how the filter works:

1. IMU angular rates are integrated to calculate the angular position
2. IMU accelerations are converted using the angular position from body X,Y,Z to earth North,East and Down axes and corrected for gravity
3. Accelerations are integrated to calculate the velocity
4. Velocity is integrated to calculate the position

This process from 1) to 4) is referred to as 'State Prediction'. A 'state' is a variables we are trying to estimate like roll,pitch yaw, height, wind speed, etc. The filter has other states besides position, velocity and angles that are assumed to change slowly. These include gyro biases, Z accelerometer bias, wind velocities, compass biases and the earth's magnetic field. These other states aren't modified directly by the 'State Prediction' step but can be modified by measurements a described later.

5. Estimated gyro and accelerometer noise (`EKF_GYRO_NOISE` and `EKF_ACC_NOISE`) are used to estimate the growth in error in the angles, velocities and position calculated using IMU data. Making these

parameters larger causes the filters error estimate to grow faster. If no corrections are made using other measurements (eg GPS), this error estimate will continue to grow. These estimated errors are captured in a large matrix called the 'State Covariance Matrix'.

Steps 1) to 5) are repeated every time we get new IMU data until a new measurement from another sensor is available.

If we had a perfect initial estimate, perfect IMU measurements and perfect calculations, then we could keep repeating 1) to 4) throughout the flight with no other calculations required. However, errors in the initial values, errors in the IMU measurements and rounding errors in our calculations mean that we can only go for a few seconds before the velocity and position errors become too large.

The Extended Kalman Filter algorithm provides us with a way of combining or fusing data from the IMU, GPS, compass, airspeed, barometer and other sensors to calculate a more accurate and reliable estimate of our position, velocity and angular orientation.

The following example describes how GPS horizontal position measurements are used, however the same principal applies to other measurement types (barometric altitude, GPS velocity, etc)

6. When a GPS measurement arrives, the filter calculates the difference between the predicted position from 4) and the position from the GPS. This difference is called an 'Innovation'.
7. The 'Innovation' from 6), 'State Covariance Matrix' from 5), and the GPS measurement error specified by `EKF_POSNE_NOISE` are combined to calculate a correction to each of the filter states. This is referred to as a 'State Correction'.

This is the clever part of the a Kalman Filter, as it is able to use knowledge of the correlation between different errors and different states to correct states other than the one being measured. For example GPS position measurements are able to correct errors in position, velocity, angles and gyro bias.

The amount of correction is controlled by the assumed ratio of the error in the states to the error in the measurements. This means if the filter thinks its own calculated position is more accurate than the GPS measurement, then the correction from the GPS measurement will be smaller. If it thinks its own calculated position is less accurate than the GPS measurement, then the correction from the GPS measurement will be larger. The assumed accuracy of the GPS measurement is controlled by the `EKF_POSNE_NOISE`, parameter. Making `EKF_POSNE_NOISE` larger causes the filter to think the GPS position is less accurate.

8. Because we have now taken a measurement, the amount of uncertainty in each of the states that have been updated is reduced. The filter calculates the reduction in uncertainty due to the 'State Correction', updates the 'State Covariance Matrix' and returns to step 1)

## Tuning Parameters

### **AHRS\_EKF\_USE**

This should be set to 1 to enable use of the filter, or set to 0 to use the legacy algorithms. Be aware that both algorithms are running regardless of this parameter, and all the EKF data will be logged regardless, provided full rate AHRS data logging is enabled.

From Copter3.3 onwards the EKF has been enabled by default and this parameter is not available. Plane and Rover users can still elect to use the legacy algorithms.

### **EKF\_ABIAS\_PNOISE**

This noise controls the growth of the vertical accelerometer bias state error estimate. Increasing it makes accelerometer bias estimation faster and noisier.

### **EKF\_ACC\_PNOISE**

This noise controls the growth of estimated error due to accelerometer measurement errors excluding bias. Increasing it makes the filter trust the accelerometer measurements less and other measurements more.

### **EKF\_ALT\_NOISE**

This is the RMS value of noise in the altitude measurement. If you increase this parameter, the filter will think the barometer is more noisy and will place less weighting on its measurements.

If this parameter is set too small, then the filter will constantly react to noise in the barometer measurement which will cause the filter height to be noisy. In copters this will cause the copter to jiggle up and down during altitude hold.

If this parameter is set too high, then the height will tend to wander more and will be more susceptible to GPS vertical velocity glitches.

See the section on interpreting EKF3 log data for more information on using log data to help set this parameter.

### **EKF\_ALT\_SOURCE**

This parameter controls which measurement source is used to determine height during optical flow navigation. Set to 0 to use the barometer or to 1 to use the range finder. If set to 1, the vehicle will attempt to maintain a constant height relative to the terrain, which is the default behaviour during optical flow navigation.

### **EKF\_EAS\_GATE**

This parameter scales the threshold used for the airspeed measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted. It is scaled in units of standard deviation. For example a setting of 3 means that differences greater than than  $3 \times$  the assumed standard deviation will cause the measurement to be rejected.

### **EKF\_EAS\_NOISE**

This is the RMS value of noise in compass measurements. Increasing it reduces the weighting on these measurements. See the section on interpreting EKF3 log data for more information on using log data to help set this parameter. See the section on interpreting EKF3 log data for more information on using log data to help set this parameter.

### **EKF\_FALLBACK**

This parameter controls whether inconsistency in sensor data can cause a fallback to DCM. If set to 0, then detection of inconsistent sensor cannot cause a fallback. If set to 1, then large inconsistencies in data will result a fallback to DCM if available.

### **EKF\_FLOW\_DELAY**

This is the number of msec that the optical flow rate measurements lag behind the IMU measurements.

### **EKF\_FLOW\_GATE**

This parameter controls the maximum amount of difference in between the measured optical flow rates and the predicted rates before the EKF starts to reject the measurements. Reducing this parameter makes it more likely that valid optical flow rate measurements will be rejected. Increasing this parameter makes it more likely that invalid optical flow rate measurements will be accepted. It is scaled in units of standard deviation. For example a setting of 3 means that differences greater than than  $3 \times$  the assumed standard deviation will cause the measurement to be rejected.

#### **EKF\_FLOW\_NOISE**

This parameter allows for optical flow rate measurement errors and noise. It represents the expected RMS error in rad/sec. If set too large the position will drift more. If set too small the position and velocity output from the EKF will become noisy and there is a risk that the EKF could start rejecting optical flow measurements during manoeuvres.

#### **EKF\_GBIAS\_PNOISE**

This noise controls the growth of gyro bias state error estimates. Increasing it makes rate gyro bias estimation faster and noisier.

#### **EKF\_GLITCH\_ACCEL**

This parameter controls the maximum amount of difference in horizontal acceleration (in cm/s<sup>2</sup>) between the value predicted by the filter and the value measured by the GPS, before the GPS position measurement is rejected. If this value is set too low, then valid GPS data will be regularly discarded, and the position accuracy will degrade. If this parameter is set too high, then GPS glitches can cause large rapid changes in position.

#### **EKF\_GLITCH\_RAD**

This parameter controls the maximum amount of difference in horizontal position (in m) between the value predicted by the filter and the value measured by the GPS, before the long term glitch protection logic is activated and an offset is applied to the GPS measurement to compensate. Position jumps smaller than this parameter will be temporarily ignored, but if they persist will then be accepted and the filter will move to the new position. Position steps larger than this value, will also be ignored initially, but if they persist, the GPS position measurement will be corrected by the amount of the step before being used. This prevents a large step change in position. This correction is decayed back to zero at a constant rate so that the new GPS position will be realised gradually. The value of this correction in the north and east directions can be checked by plotting the EKF4.OFN and EKF4.OFE flashlog data.

#### **EKF\_GND\_GRADIENT**

This parameter controls the amount of terrain gradient in % that is assumed when fusing range finder data and influences how rapidly the estimated terrain height responds to changes in measurement. This can be increased when operating over uneven terrain to allow the terrain estimate to change more rapidly.

#### **EKF\_GPS\_TYPE**

This parameter controls use of GPS velocity measurements : 0 = use 3D velocity, 1 = use 2D velocity, 2 = use no velocity

#### **EKF\_GYRO\_PNOISE**

This noise controls the growth of estimated error due to gyro measurement errors excluding bias.

Increasing it makes the filter trust the gyro measurements less and other measurements more.

### **EKF\_HGT\_GATE**

This parameter scales the threshold used for the height measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

### **EKF\_MAGB\_PNOISE**

This noise controls the growth of body magnetic field state error estimates. Increasing it makes compass offset estimation faster and noisier.

### **EKF\_MAGE\_PNOISE**

This noise controls the growth of earth magnetic field state error estimates. Increasing it makes earth magnetic field bias estimation faster and noisier.

### **EKF\_MAG\_CAL**

The EKF is capable of learning magnetometer offsets in-flight. This parameter controls when the learning is active:

- `EKF_MAG_CAL = 0`: Learning is enabled when speed and height indicate the vehicle is airborne
- `EKF_MAG_CAL = 1`: Learning is enabled when the vehicle is manoeuvring
- `EKF_MAG_CAL = 2`: Learning is disabled
- `EKF_MAG_CAL = 3`: Learning is enabled when the vehicle is armed

### **EKF\_MAG\_GATE**

This parameter scales the threshold used for the magnetometer measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted. It is scaled in units of standard deviation. For example a setting of 3 means that differences greater than than  $3 \times$  the assumed standard deviation will cause the measurement to be rejected.

### **EKF\_MAG\_NOISE**

This is the RMS value of noise in magnetometer measurements / 1000. The magnetometer readings are scaled by 1/1000 before they are used by the filter to reduce the effect of numerical rounding errors. Increasing this noise parameter reduces the weighting on magnetometer measurements. This would make the filter yaw less affected less by magnetometer errors, but more affected by Z gyro drift. See the section on interpreting EKF3 log data for more information on using log data to help set this parameter.

### **EKF\_MAX\_FLOW**

This parameter controls the maximum amount of optical flow rate (in rad/sec) that will be accepted as a valid measurement by the EKF. This helps to reject measurements corrupted during data transfer or when the flow sensor is unable to keep up with the motion of the vehicle.

### **EKF\_POS\_DELAY**

This is the number of msec that the GPS position measurements lag behind the inertial measurements.

**EKF\_POSNE\_NOISE**

This is the RMS value of noise in the GPS horizontal position measurements. If you increase this parameter, the filter will think the GPS is more noisy and will place less weighting on the horizontal GPS velocity measurements.

If this parameter is set to small, then the filter will constantly react to noise in the GPS position which can cause continual and rapid small attitude and position changes in copters during loiter.

If this parameter is set to large, then the inertial sensor errors will cause the filter position to wander slowly as errors in the inertial calculations are not corrected enough by the GPS. This can cause excessive wander in position for copters during loiter.

See the section on interpreting EKF3 log data for more information on using log data to help set this parameter.

**EKF\_POS\_GATE**

This parameter scales the threshold used for the GPS position measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted. It is scaled in units of standard deviation. For example a setting of 3 means that differences greater than than  $3 \times$  the assumed standard deviation will cause the measurement to be rejected.

**EKF\_RNG\_GATE**

This parameter controls the maximum amount of difference in between the measured range to ground and the predicted range before the EKF starts to reject the measurements. Reducing this parameter makes it more likely that valid range finder measurements will be rejected. Increasing this parameter makes it more likely that invalid range finder measurements will be accepted. It is scaled in units of standard deviation. For example a setting of 3 means that differences greater than than  $3 \times$  the assumed standard deviation will cause the measurement to be rejected.

**EKF\_VELD\_NOISE**

This is the RMS value of noise in the vertical GPS velocity measurement in m/s. If you increase this parameter, the filter will think the GPS is more noisy and will place less weighting on the vertical GPS velocity measurements.

If this parameter is set too small, then the filter will constantly react to noise in the GPS measurement which will cause the filter height to be noisy. In copters this will cause the copter to jiggle up and down.

If this parameter is set too high then the filter will not take full advantage of the GPS velocity information, and will be more susceptible to Barometer height glitches.

See the section on interpreting EKF3 log data for more information on using log data to help set this parameter.

**EKF\_VELNE\_NOISE**

This is the RMS value of noise in the North and East GPS velocity measurements in m/s. If you increase this parameter, the filter will think the GPS is more noisy and will place less weighting on the horizontal GPS velocity measurements.

If this parameter is set too small, then the filter will constantly react to noise in the GPS measurement

which will cause the filter roll and pitch angles to be noisy. If you have the vehicle outside with a clear view of the sky and away from buildings and other large objects, then the HUD in mission planer should be steady. If it is moving around noticeably, then it is likely the GPS noise is too high for the filter setting. This will also result in continual and rapid small angle and position changes in copters during loiter.

If this parameter is set too high then the filter will not take full advantage of the GPS velocity information, will wander more in position and will be more susceptible to GPS position glitches.

See the section on interpreting EKF3 log data for more information on using log data to help set this parameter.

### **EKF\_VEL\_DELAY**

This is the number of msec that the GPS velocity measurements lag behind the inertial measurements.

### **EKF\_VEL\_GATE**

This parameter scales the threshold used for the GPS velocity measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted. It is scaled in units of standard deviation. For example a setting of 3 means that differences greater than than  $3 \times$  the assumed standard deviation will cause the measurement to be rejected.

### **EKF\_WIND\_PNOISE**

This noise controls the growth of wind state error estimates. Increasing it makes wind estimation faster and noisier.

### **EKF\_WIND\_PSCALE**

Increasing this parameter increases how rapidly the wind states adapt when changing altitude, but does make wind speed estimation noisier.

## **Interpreting Log Data**

Correct tuning the Navigation filter is not possible without some analysis of the data logged by the filter in the flash logs. To log this data, it is important that AHRS data logging is enabled. The EKF data is contained in the EKF1, EKF2, EKF3 and EKF4 log messages. This section describes the meaning of the various EKF log data and shows examples obtained from plotting data using the Mission Planner DataFlash log review feature.

### **EKF1**

**TimeMS** - time in msec from startup

**Roll** - Roll angle (deg)

**Pitch** - Pitch angle (deg)

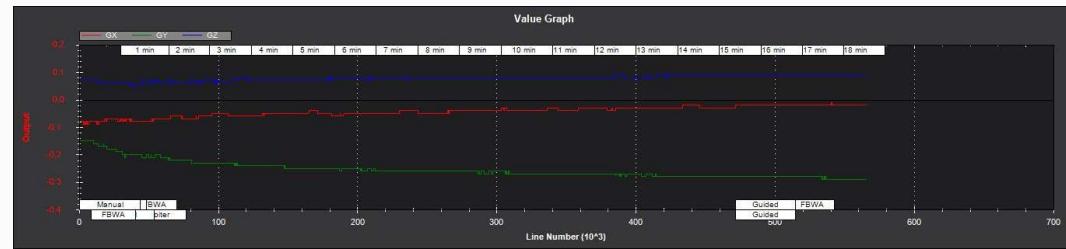
**Yaw** - Yaw angle (deg)

**VN,VE,VD** - North,East,Down velocities (m/s)

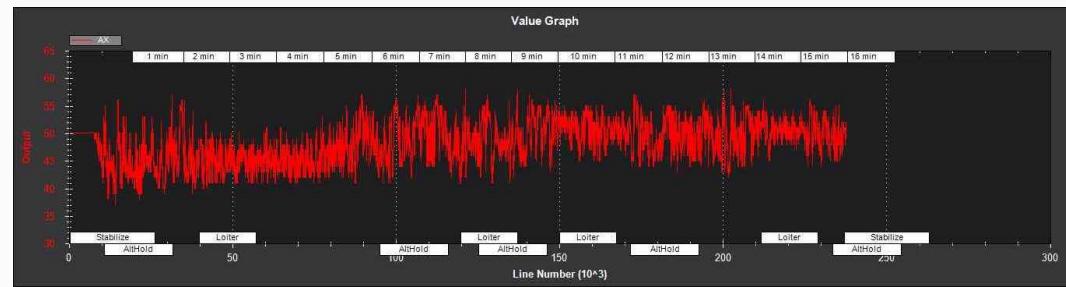
**PN,PE,PD** - North,East,Down positions (m) relative to where the vehicle was armed

**GX,GY,GZ - X,Y,Z Gyro biases (deg/min)**

The following figure shows the gyro biases from a plane with a Pixhawk controller. The gyro biases can be seen to vary at the start and stabilise about new values as the sensor warms up and reaches its operating temperature. The cheap MEMS inertial sensors used by our controllers can have significant bias variation with temperature.

**EKF2****TimeMS** - time in msec from startup.

**Ratio** - Weighting percentage of the IMU1 accelerometer data used in the blending of IMU1 and IMU2 data. If two IMU's are available with your hardware (eg Pixhawk), then this will normally fluctuate rapidly in the 50% region as seen here.

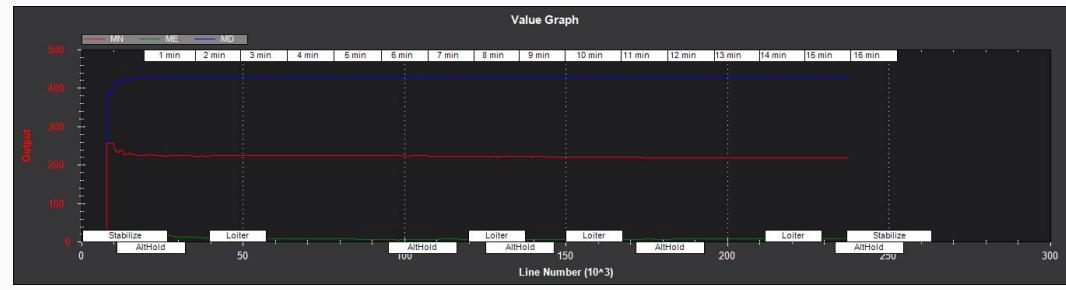


If it swings close to 100 or 0 % for parts of the flight, then this indicates that you likely have aliasing affecting your accelerometer data and you should look for solutions to reduce this (eg vibration isolation mounts for your autopilot).

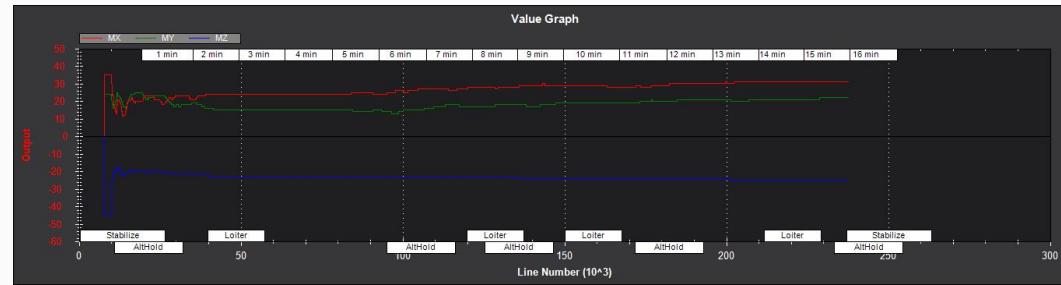
**AZ1bias** - Z accelerometer bias for IMU1 (cm/s:sup:2)**AZ2bias** - Z accelerometer bias for IMU2 (cm/s:sup:2)

**VWN,VWE** - North and East wind velocity (m/s). A positive value means the wind is moving in the direction of that axis, eg a positive North wind velocity is blowing from the South.

**MN,ME,MD** - North, East, Down earth magnetic field strength (sensor units). If you are flying quickly, or are at low speed with **EKF\_MAG\_CAL** enabled, these will slowly change during flight as the filter 'learns' the earth's magnetic field.



**MX,MY,MZ** - X, Y, Z body magnetic field biases (sensor units). If you are flying quickly, or are at low speed with `EKF_MAG_CAL` enabled, these will slowly change during flight as the filter 'learns' the earth's magnetic field. These have the same meaning as the compass offsets, but are the opposite sign (eg in the following figure MX stabilises at a value of +35, indicating that a `COMPASS_OFS_X` value of -35 should be used.

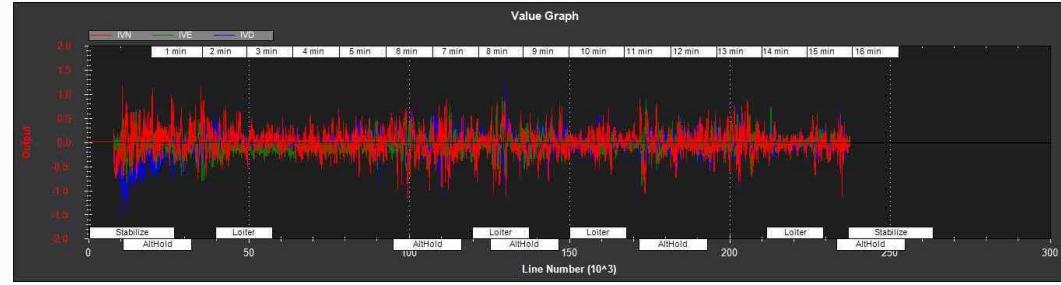


### EKF3

This message contains the innovations for each sensor (GPS, barometer, magnetometer and airspeed). Innovations are the difference between the value predicted using the IMU data before corrections are applied, and the value measured by the sensor.

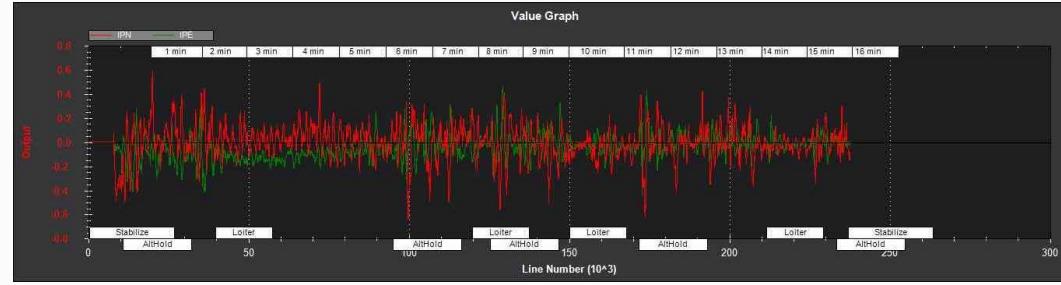
**TimeMS** - Time in msec from startup

**IVN,IVE,IVD** - Innovations for the North,East,Down GPS velocity measurements (m/s). These are an important measure of health for the navigation filter. If you have god quality IMU and GPS data they will be small and around zero as shown in the following figure:



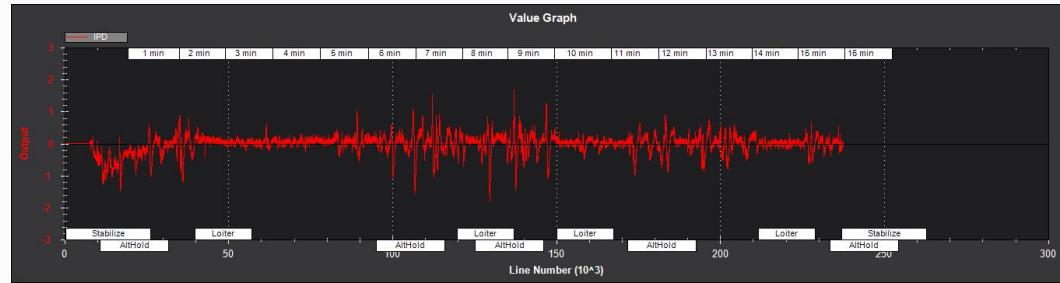
The noise level on these innovations when the vehicle is not maneuvering can be used to set the value of `EKF_VELNE_NOISE` and `EKF_VELD_NOISE`. For example in the above figure, the velocity noise when the vehicle was non-manoeuvring was around +-0.3 m/s for both the North,East and Down velocities. This means that a good starting value for `EKF_VELNE_NOISE` and `EKF_VELD_NOISE` for this example would be 0.3 m/s.

**IPN,IPE** - Innovations in the North, East GPS position measurements (m). Similarly to the velocity innovations, they should be small and centred on zero as in the following example:



The noise levels on these innovations can be used to set the value of `EKF_POSNE_NOISE`. In the above figure, the noise sits within a band of +-0.5m, so a good starting value for the value of `EKF_POSNE_NOISE` in this example would be 0.5m.

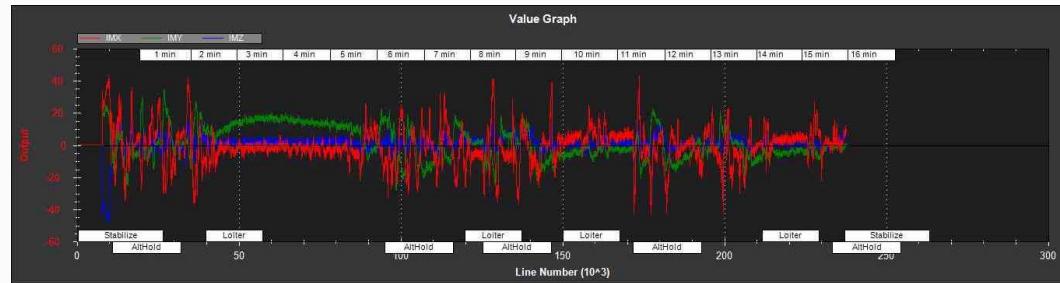
**IPD** - Innovations on the barometer height measurement (m). They should be small and centered on zero as in the following example, although transients of around 2m are common when sudden height changes or manoeuvres are performed due to IMU errors, sensor lag and the effect of changes in airflow on the barometer reading.



In the above figure it can be seen that there is a small 1m negative offset that is removed after 2min. This is due to bias errors on the Z accelerometers which take time to be learned by the filter and compensated for. In this example, the underlying sensor noise is low at about  $\pm 0.15$ m, which indicates a good starting value for `EKF_ALT_NOISE` for plane applications would be 0.15m.

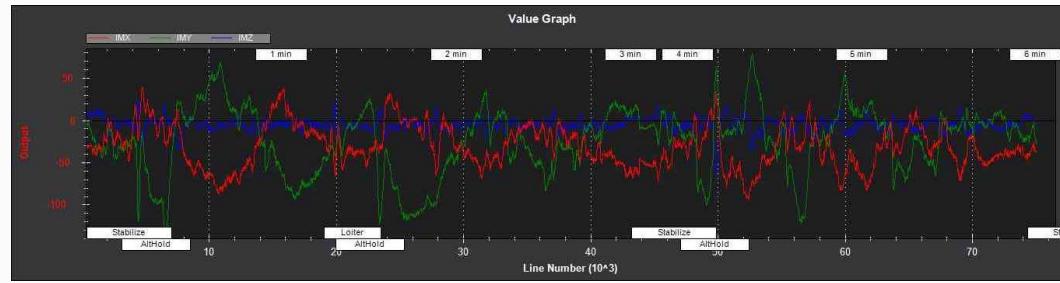
Note: For copter, experience has shown the value of `EKF_ALT_NOISE` normally has to be increased above the theoretical value to smooth out the height response

**IMX,IMY,IMZ** - Innovations for the Magnetometer X,Y,Z measurements. These should be centered around zero and not exceed  $\pm 50$  during manoeuvres as shown in the following figure:



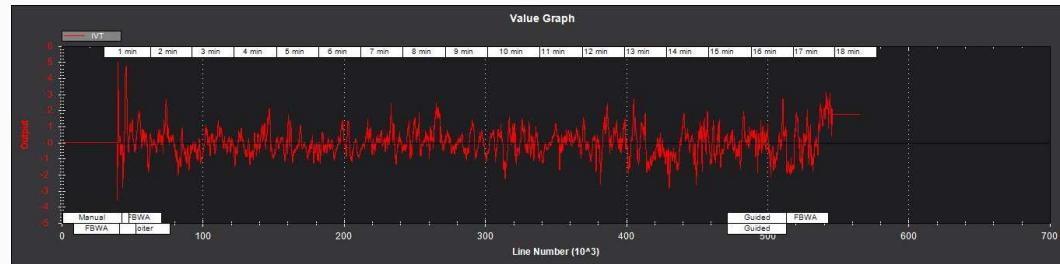
In the above example `EKF_MAG_CAL` was set to 1, so the copter quickly learnt the magnetometer biases (compass offsets). Although the underlying noise of the magnetometer is relatively low (5 or less in most cases), there are other errors due to differences in scale factors between axes, magnetometer misalignment, and varying magnetic fields produced by electrical power systems that cause larger errors. Typically these result in sharp transients of about 50 in the innovations, as can be seen in the above figure. For this reason the default value of `EKF_MAG_NOISE` is set to 0.05 (which represents a noise of 50 in sensor units).

The following figure is taken from a slow speed copter flight with a bad magnetometer calibration and `EKF_MAG_CAL` = 0. The innovations vary noticeably as the vehicle changes its orientation.



**IVT** - Innovation for the true airspeed measurement (m/s). This will be zero if the airspeed sensor is not

fitted or is not being used (e.g. on ground). It should be centered around zero if the airspeed sensor is calibrated correctly, but will vary in noise level depending on how gusty the flight conditions are. The following is an example from a flight with a well calibrated airspeed sensor in moderate wind conditions of around 7m/s in low turbulence:



A constant offset of 1m/s from zero would indicate a steady 1m/s airspeed error. Steady airspeed errors can be caused if the airspeed sensor is uncovered during initialisation on a windy day resulting in a significant pressure offset, is out of cal, or has experienced a large change in temperature since initialisation.

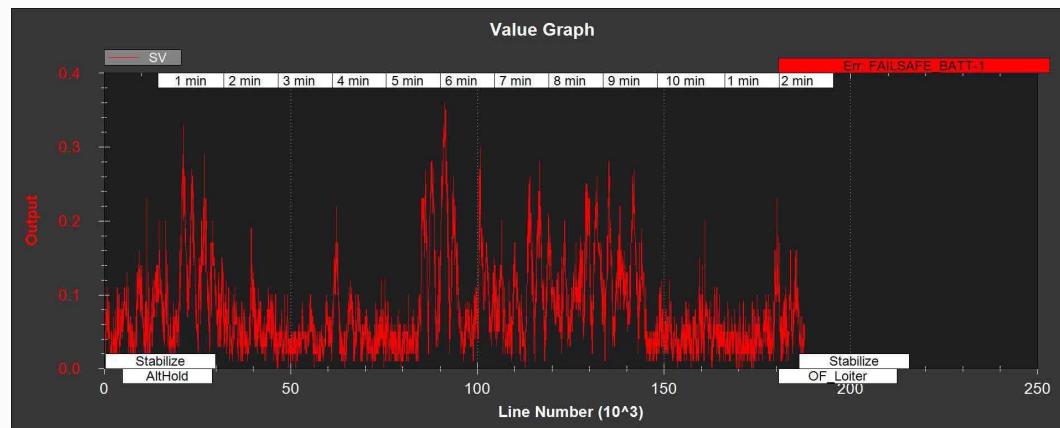
This figure can also be used to set the value for `EKF_EAS_NOISE`. For the example shown above, the total noise (including gusts) is around 1.4 m/s, so this would be a good starting value for `EKF_EAS_NOISE`.

## EKF4

This message contains plots showing how each sensor is performing relative to the error gates set by the `EKF_POS_GATE`, `EKF_VEL_GATE`, `EKF_HGT_GATE`, `EKF_MAG_GATE` and `EKF_EAS_GATE`. These parameters control how inconsistent a measurement is allowed to be before the filter won't use it. When we refer to inconsistency of measurements in this section, we are talking about the amount of difference between the measurement predicted by the filter and the measurement taken by the sensor. Checking measurements for inconsistencies is particularly important with GPS, because GPS measurements can have very large transient position and velocity errors that would cause a crash if they were to be used by the filter. The following messages are available in EKF4:

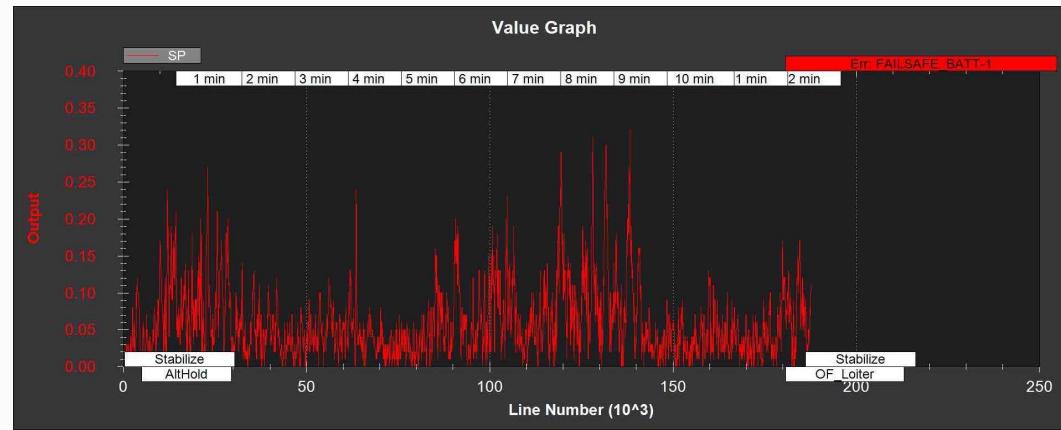
**TimeMS** - Time in msec from startup

**SV** - ratio of the combined GPS velocity inconsistency to the limit set by the `EKF_VEL_GATE` parameter. For a flight with good GPS data, this can have the occasional spike to over 1/2, but should never go above 1. If this line goes above 1, then it indicates that the filter stopped using the GPS velocity data for that period in flight. This should never happen with good sensor data. The following figure shows **SV** taken from a quadrotor flight with 9 to 10 satellites in good GPS conditions, using the default parameters. If this line is too high and goes above 1 with good GPS, then the `EKF_VEL_GATE` parameter should be increased.

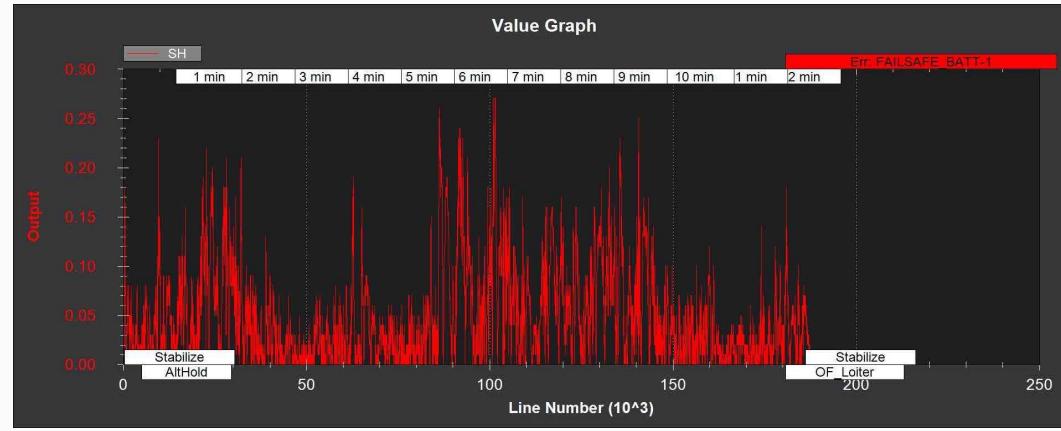


**SP** - ratio of the GPS total position inconsistency to the limit set by the `EKF_POS_GATE` parameter. For a

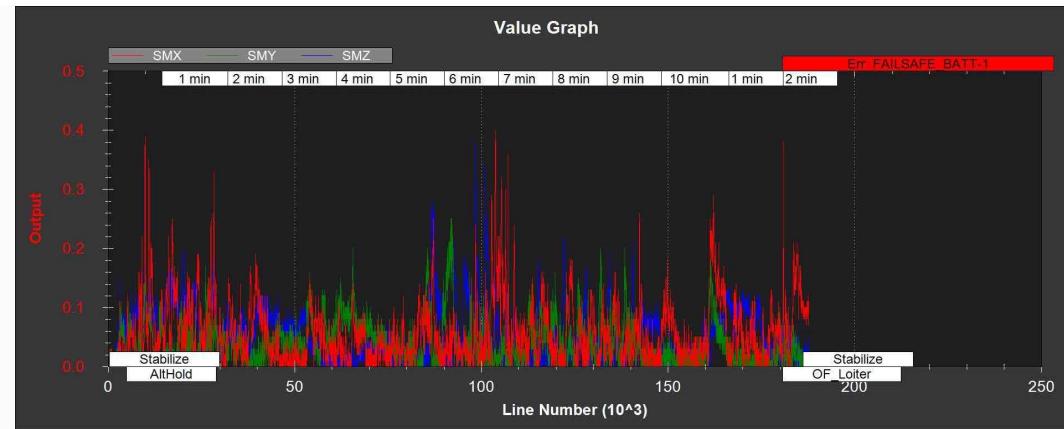
flight with good GPS data, this can have the occasional spike to over 1/2, but should never go above 1. If this line goes above 1, then it indicates that the filter stopped using the GPS position data for that period in flight. This should never happen with good sensor data. The following figure shows **SP** taken from a quadrotor flight with 9 to 10 satellites in good GPS conditions, using the default parameters. If this line is too high and goes above 1 with good GPS, then the `EKF_POS_GATE` parameter should be increased.



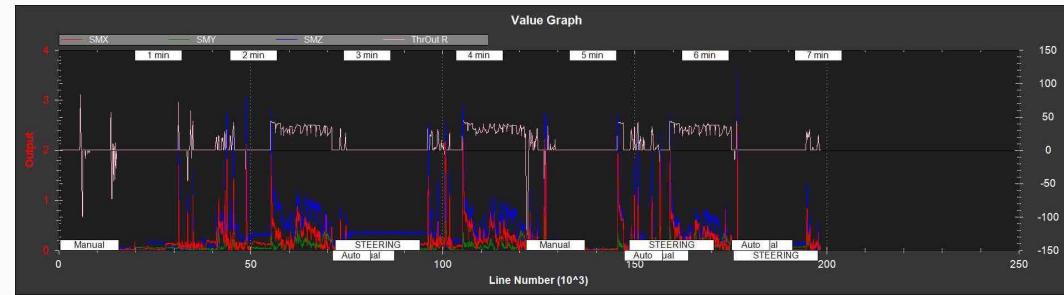
**SH** - ratio of the barometer height inconsistency to the limit set by the `EKF_HGT_GATE` parameter. This can have the occasional spike to over 1/2, but should never go above 1. If this line goes above 1, then it indicates that the filter stopped using the barometer data for that period in flight. This should never happen with good sensor data. The following figure shows **SH** taken from a quadrotor flight at airspeeds up to 16 m/s, using the default parameters. If this line is too high and goes above 1, then the ``EKF\_HGT\_GATE`` parameter should be increased. Factors that can cause this to be high include airflow past the autopilot affecting the barometer reading and accelerometer errors due to sensor drift or aliasing.



**SMX,SMY,SMZ** - ratio of the magnetometer X,Y and Z measurement inconsistencies to the limit set by the `EKF_MAG_GATE` parameter. This can have the occasional spike to over 1/2, but should never go above 1. If this line goes above 1, then it indicates that the filter stopped using that component of magnetometer data for that period in flight. This should never happen with good sensor data. The following figure shows the SMX, SMY and SMZ data taken from a quadrotor flight using the default parameters. If this line is too high and goes above 1 on a regular basis, then it indicates a problem with the compass calibration or installation. It is recommended that the reasons for the compass errors be investigated first before resorting to increasing the `EKF_MAG_GATE` parameter.

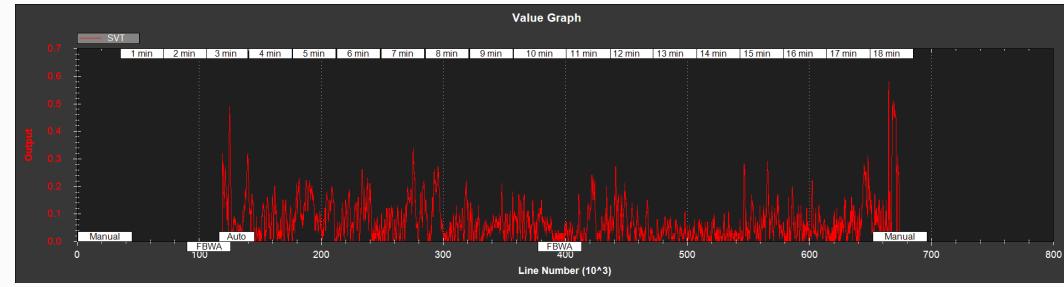


The next figure shows **SMX**, **SMY** and **SMZ** taken from a rover log, with the throttle demand **ThrOutR** also plotted.



The large spikes above 1 every time the throttle steps up, and the large values during throttle operation can be clearly seen. In this example it would be recommended that steps be taken to reduce the amount of compass interference

**SVT** - ratio of the airspeed measurement inconsistency to the limit set by the `EKF_EAS_GATE` parameter. This can have the occasional spike to over 1/2, but should rarely go above 1. If this line goes above 1, then it indicates that the filter stopped using the airspeed data for that period in flight. Factors that can cause this to be high include airspeed calibration errors, the presence of strong gusts and turbulence, and rapid changes in wind speed. It is normal for this to be higher at the start of the flight before the filter has estimated the wind velocity.



## EKF2 Navigation System

### Warning

EKF2 is currently in master only. This topic is provided for advanced users only.

### Overview

### What is the EKF2 Navigation System?

It is a 24 state extended Kalman filter in the AP\_NavEKF2 library that estimates the following states

- Attitude (Quaternions)
- Velocity (North,East,Down)
- Position (North,East,Down)
- Gyro bias offsets (X,Y,Z)
- Gyro scale factors (X,Y,Z)
- Z accel bias
- Earth magnetic field (North,East,Down)
- Body magnetic field (X,Y,Z)
- Wind Velocity (North,East)

It is based on the filter equations derived here:

<https://github.com/priseborough/InertialNav/blob/master/derivations/RotationVectorAttitudeParameterisation/GenerateNavFilterEquations.m>

## EKF2 Advantages

- It can run a separate EKF2 for each IMU making recovery from an IMU fault much more likely
- It can switch between magnetometers if there is fault
- It can estimate gyro scale factors which can improve accuracy during high rate manoeuvres
- It can simultaneously estimate both gyro offsets and orientation on startup whilst moving and doesn't rely on the DCM algorithm for its initial orientation. This makes it ideal for flying from moving platforms when the gyro has not been calibrated.
- It can handle larger gyro bias changes in flight
- It is able to recover faster from bad sensor data
- It provides a slightly smoother output.
- It is slightly more accurate
- It uses slightly less computing power
- It starts using GPS when checks pass rather than waiting for the vehicle motors to arm.

## How does it achieve this ?

1. Instead of trying to estimate the quaternion orientation directly, it estimates an error rotation vector and applies the correction to the quaternion from the inertial navigation equations. This is better when there are large angle errors as it avoids errors associated with linearising quaternions across large angle changes.
  - See "Rotation Vector in Attitude Estimation", Mark E. Pittelkau, Journal of Guidance, Control, and Dynamics, 2003" for details on this approach.
2. The new EKF runs on a delayed time horizon so that when measurements are fused, they are done so using the measurement, filter states and covariance matrix from the same point in time. The legacy EKF uses a simpler method where delayed measurements are fused using delayed states but the covariance matrix is from the current time which reduces accuracy.
  - The delayed filter states are then predicted forward into the current time horizon using a complementary filter that removes the time delay and also filters out the sudden changes in the states that occur when measurements are fused.
  - This approach was inspired by the output predictor work done by Ali Khosravian from ANU. "Recursive Attitude Estimation in the Presence of Multi-rate and Multi-delay Vector Measurements," A Khosravian, J Trumpf, R Mahony, T Hamel - American Control Conference, vol.-, 2015.
  - This is computationally much cheaper than winding the states forward using buffered IMU data at each update, but slightly less accurate. The legacy EKF smooths the state corrections by applying them incrementally across the time to the next measurement which reduces stability of the filter.
3. Further optimisation of mathematics and code to improve speed have been introduced.
4. A simple compass yaw angle fusion method with fixed declination has been added which is used on the ground or when magnetic interference prevents use of the more accurate 3-axis 6-state magnetometer fusion technique.

## Using EKF2

1. Enable the new EKF by setting EK2\_ENABLE = 1. EKF2 will now be running in parallel and logging, etc but it will not be used by the control loops.
2. To use it in the control loops, set AHRS\_EKF\_TYPE = 2
3. The use of multiple IMU's, is controlled by the EK2\_IMU\_MASK parameter:
  - To use only IMU1, set EK2\_IMU\_MASK = 1 (this is the default)
  - To use only IMU2, set EK2\_IMU\_MASK = 2
  - To run dual EKF2 using IMU1 and IMU2, set EK2\_IMU\_MASK = 3 and turn off the legacy EKF by setting EKF\_ENABLE = 0
4. After setting the parameters you will need to reboot.

## EKF2 Log Data

The data for EKF2 can be found in the NKF1 to NKF9 log packets.

Packets NKF1 to NKF 4 contain information for the first EKF instance. Packets NKF6 to NKF9 contain the same information, for the second instance if enabled using the EK2\_IMU\_MASK parameter. Packet NKF5 contains optical flow information for the EKF instance that is the primary for flight control.

The available EKF2 available log data is listed below. Some plots showing example flight data have been included. This data was logged using a Pixhawk on a 3DR Iris+ quadcopter with the following parameter changes followed by a reboot:

- EKF\_ENABLE = 0 (turns off the legacy EKF)
- EK2\_ENABLE = 1 (turns on EKF2)
- EK2\_IMU\_MASK = 3 (Instructs EKF2 to run two instances, one for IMU1 (MPU6000) and one for IMU2 (LDG20H + LSM303D))
- AHRS\_EKF\_TYPE = 2 (tells the flight control system to use EKF2)
- LOG\_BITMASK = 131071 (logs 50Hz data from startup)

**\*Note: Plots can be viewed in full size by left clicking them\***

## Filter State Estimates

**NKF1** (and **NKF6** if a second IMU is being used) contain the outputs used by the flight control system

- TimeUS - time stamp (uSec)
- Roll,Pitch - Euler roll and pitch angle (deg)
  - [?](#)
- Yaw - Euler yaw angle (deg)
  - [?](#)
- VN,VE - Velocity North,East direction (m/s)
  - [?](#)
- VD, dPD - Velocity and Position Derivative Down (m/s)
  - [?](#)
- PN,PE,PD - Position North,East,Down (m)
  - [?](#)
- GX,GY,GZ - X,Y,Z rate gyro bias (deg/sec)
  - [?](#)

**NKF2** (and **NKF7** if a second IMU is being used) contains additional state information

- TimeUS - time stamp (uSec)
- AZbias - Z accelerometer bias (cm/s/s)



- GSX,GSY,GSZ - X,Y,Z rate gyro scale factor (%)

Eg, a log value of 0.5 would be equivalent to a scale factor of 1.005 for that sensor



- VWN,VWE - Wind velocity North,East (m/s)
- MN,ME,MD - Earth magnetic field North,East,Down (mGauss)



- MX,MY,MZ - Body fixed magnetic field X,Y,Z (mGauss)



- MI - Index of the magnetometer being used by EKF2

## Filter Innovations

**NKF3** (and **NKF8** if a second IMU is being used) contain information on the filter innovations. An innovation is the difference between the measurement value predicted by EKF2 and the value returned by the sensor. Smaller innovations indicate smaller sensor errors. Because the IMU data is used to do the prediction, bad IMU data can result in large innovations for all measurements.

- TimeUS - time stamp (uSec)
- IVN,IVE - GPS velocity innovations North, East (m/s)



- IVD - GPS velocity innovation Down (m/s)



- IPN,IPE - GPS position innovations North,East (m)



- IPD - Barometer position innovation Down (m)



- IMX,IMY,IMZ - Magnetometer innovations X,Y,Z (mGauss)



- IYAW - Compass yaw innovation (deg)



- IVT - True airspeed innovation (m/s)

## Filter Health and Status

**NKF4** (and **NKF9** if a second IMU is being used) contain information on the innovation variance test ratios. A value of less than 1 indicates that that measurement has passed its checks and is being used by the EKF2. A value of more than 1 indicates that the innovations for that measurement are so high that the EKF2 will be rejecting the data from that sensor. Values of less than 0.3 in flight are typical for a setup with good quality sensor data.

They also contain other information relevant to filter health

- TimeUS - time stamp (uSec)
- SV - GPS velocity test ratio
- SP - GPS position test ratio
- SH - Barometer test ratio
- SM - Magnetometer test ratio
- SVT - Airspeed sensor Test ratio



- errRP - Estimated attitude roll/pitch error (rad)



- OFN - Position jump North due to the last reset of the filter states (m)
- OFE - Position jump East due to the last reset of the filter states (m)
- FS - Integer bitmask of filter numerical faults
- TS - Integer bitmask of filter measurement timeout
- SS - Integer bitmask of filer solution status
- GPS - Integer bitmask of filter GPS quality checks
- PI - Index showing which instance of EKF2 has been selected for flight control

## **Optical Flow and Range Finder Fusion**

**NKF5** contains information on the optical flow fusion for the EK2 instance bing used for flight control

- TimeUS - time stamp (uSec)
- normInnov - optical flow innovation variance test ratio
- FIX,FIY - optical flow X and Y axis innovations (mrad/s)



- AFI - optical flow terrain height estimator innovation (mrad/s)
- HAGL - estimated height above ground level (m)
- meaRng - Range measured by the range finder (m)



- offset - estimated terrain offset relative to the pressure height origin



- RI - Range finder innovation (m)



- errHAGL - 1-Sigma uncertainty in the terrain height offset estimate (m)



## **Tuning Parameters**

The EKF2 parameters have been tuned to provide a compromise between accuracy and robustness to sensor errors. It is likely that further improvements in performance are available with further tuning.

If you have a question regarding tuning of the filer, please post to  
<https://groups.google.com/forum/#!forum/drones-discuss> along with your log file and mention the term EKF2 in your post title.

The parameters for the new EKF start with the prefix EK2\_ and are listed below

#### **EK2\_ENABLE**

This turns the EKF 2 on and off. Set to 1 to turn on and 0 to turn off. Turning EKF2 on only makes the calculations run, it does not mean it will be used for flight control. To use it for flight control set AHRS\_EKF\_TYPE=2. A reboot or restart will need to be performed after changing the value of EK2\_ENABLE for it to take effect.

#### **EK2\_GPS\_TYPE**

This controls the use of GPS measurements :

- 0 = use 3D velocity & 2D position
- 1 = use 2D velocity and 2D position
- 2 = use 2D position
- 3 = use no GPS (optical flow will be used if available)

#### **EK2\_VELNE\_NOISE**

This sets a lower limit on the speed accuracy reported by the GPS receiver that is used to set horizontal velocity observation noise. If the model of receiver used does not provide a speed accuracy estimate, then the parameter value will be used. Increasing it reduces the weighting of the GPS horizontal velocity measurements. It has units of metres/sec

#### **EK2\_VELD\_NOISE**

This sets a lower limit on the speed accuracy reported by the GPS receiver that is used to set vertical velocity observation noise in. If the model of receiver used does not provide a speed accuracy estimate, then the parameter value will be used. Increasing it reduces the weighting of the GPS vertical velocity measurements. It has units of metres/sec.

#### **EK2\_VEL\_GATE**

This sets the number of standard deviations applied to the GPS velocity measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

#### **EK2\_POSNE\_NOISE**

This sets the GPS horizontal position observation noise. Increasing it reduces the weighting of GPS horizontal position measurements. It has units of metres

#### **EK2\_POS\_GATE**

This sets the number of standard deviations applied to the GPS position measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

#### **EK2\_GLITCH\_RAD**

This controls the maximum radial uncertainty in position between the value predicted by the filter and the value measured by the GPS before the filter position and velocity states are reset to the GPS. Making this

value larger allows the filter to ignore larger GPS glitches but also means that non-GPS errors such as IMU and compass can create a larger error in position before the filter is forced back to the GPS position. It has units of metres.

#### **EK2\_GPS\_DELAY**

This is the number of msec that the GPS measurements lag behind the inertial measurements. The maximum delay that can be compensated by the filter is 250 msec.

#### **EK2\_ALT\_SOURCE**

This parameter controls which height sensor is used by the EKF. If the selected option cannot be used, it will default to Baro as the primary height source. Setting 0 will use the baro altitude at all times. Setting 1 uses the range finder and is only available in combination with optical flow navigation (EK2\_GPS\_TYPE = 3). Setting 2 uses GPS. When height sources other than Baro are in use, the offset between the Baro height and EKF height estimate is continually updated. If a switch to Baro height needs to be made when the filter is operating, then the Baro height is corrected for the learned offset to prevent a sudden step in height estimate.

#### **EK2\_ALT\_NOISE**

This is the RMS value of noise in the altitude measurement. Increasing it reduces the weighting of the baro measurement and will make the filter respond more slowly to baro measurement errors, but will make it more sensitive to GPS and accelerometer errors. It has units of metres.

#### **EK2\_HGT\_GATE**

This sets the number of standard deviations applied to the height measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

#### **EK2\_HGT\_DELAY**

This is the number of msec that the height measurements lag behind the inertial measurements. The maximum delay that can be compensated by the filter is 250 msec.

#### **EK2\_MAG\_NOISE**

This is the RMS value of noise in magnetometer measurements. Increasing it reduces the weighting on these measurements. It has units of mGauss.

#### **EK2\_MAG\_CAL**

This determines when the filter will use the 3-axis magnetometer fusion model that estimates both earth and body fixed magnetic field states. This model is only suitable for use when the external magnetic field environment is stable.

- EKF\_MAG\_CAL = 0 enables calibration when airborne and is the default setting for Plane users.
- EKF\_MAG\_CAL = 1 enables calibration when manoeuvring.
- EKF\_MAG\_CAL = 2 prevents magnetometer calibration regardless of flight condition, is recommended if the external magnetic field is varying and is the default for rovers.
- EKF\_MAG\_CAL = 3 enables calibration when the first in-air field and yaw reset has completed and is the default for copters.
- EKF\_MAG\_CAL = 4 enables calibration all the time.

#### **EK2\_MAG\_GATE**

This parameter sets the number of standard deviations applied to the magnetometer measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

#### **EK2\_EAS\_NOISE**

This is the RMS value of noise in equivalent airspeed measurements used by planes. Increasing it reduces the weighting of airspeed measurements and will make wind speed estimates less noisy and slower to converge. Increasing also increases navigation errors when dead-reckoning without GPS measurements. It has units of metres/sec.

#### **EK2\_EAS\_GATE**

This sets the number of standard deviations applied to the airspeed measurement innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

#### **EK2\_RNG\_NOISE**

This is the RMS value of noise in the range finder measurement. Increasing it reduces the weighting on this measurement. It has units of metres.

#### **EK2\_RNG\_GATE**

This sets the number of standard deviations applied to the range finder innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

#### **EK2\_MAX\_FLOW**

This parameter sets the magnitude maximum optical flow rate in that will be accepted by the filter. It has units of rad/sec.

#### **EK2\_FLOW\_NOISE**

This is the RMS value of noise and errors in optical flow measurements. Increasing it reduces the weighting on these measurements. It has units of rad/sec.

#### **EK2\_FLOW\_GATE**

This sets the number of standard deviations applied to the optical flow innovation consistency check. Decreasing it makes it more likely that good measurements will be rejected. Increasing it makes it more likely that bad measurements will be accepted.

#### **EK2\_FLOW\_DELAY**

This is the number of msec that the optical flow measurements lag behind the inertial measurements. It is the time from the end of the optical flow averaging period and does not include the time delay due to the 100msec of averaging within the flow sensor.

#### **EK2\_GYRO\_PNOISE**

This control disturbance noise controls the growth of estimated error due to gyro measurement errors excluding bias. Increasing it makes the filter trust the gyro measurements less and other measurements more. It has units of rad/sec.

#### **EK2\_ACC\_PNOISE**

This control disturbance noise controls the growth of estimated error due to accelerometer measurement errors excluding bias. Increasing it makes the filter trust the accelerometer measurements less and other measurements more. It has units of metres/sec/sec.

#### **EK2\_GBIAS\_PNOISE**

This state process noise controls the growth of the gyro delta angle bias state error estimates. Increasing it makes rate gyro bias estimation faster and noisier. It has units of rad/sec.

#### **EKF2\_GSCL\_PNOISE**

This state process noise controls the rate of gyro scale factor learning. Increasing it makes rate gyro scale factor estimation faster and noisier.

#### **EK2\_ABIAS\_PNOISE**

This state process noise controls the growth of the vertical accelerometer delta velocity bias state error estimate. Increasing it makes accelerometer bias estimation faster and noisier. It has units of metres/sec/sec.

#### **EK2\_MAG\_PNOISE**

This state process noise controls the growth of magnetic field state error estimates. Increasing it makes magnetic field bias estimation faster and noisier. It has units of Gauss/sec.

#### **EK2\_WIND\_PNOISE**

This state process noise controls the growth of wind state error estimates. Increasing it makes wind estimation faster and noisier. It has units of metres/sec/sec

#### **EK2\_WIND\_PSCALE**

This controls how much the process noise on the wind states is increased when gaining or losing altitude to take into account changes in wind speed and direction with altitude. Increasing this parameter increases how rapidly the wind states adapt when changing altitude, but does make wind velocity estimates noisier.

#### **EK2\_GPS\_CHECK**

This is a 1 byte bitmap controlling which GPS preflight checks are performed. Set to 0 to bypass all checks. Set to 255 perform all checks. Set to 3 to check just the number of satellites and HDOP. Set to 31 for the most rigorous checks that will still allow checks to pass when the copter is moving, eg launch from a boat. Setting a 1 in the following bit locations causes the corresponding checks to be performed.

0: The receivers reported number of satellites must be  $\geq 6$

1: The receivers reported HDOP must be  $\geq 2.5$

2: The receivers reported speed accuracy must be less than 1.0 metres/sec (if available)

3: The receivers reported horizontal position accuracy must be less than 5.0 metres (if available)

4: The EKF2 magnetometer or compass innovation consistency checks must be passing. If these checks are failing, then the yaw estimate is unreliable

5: The rate of drift in the receivers reported horizontal position must be less than 0.3 metres/sec

6: The receivers reported vertical speed after filtering must be less than 0.3 metres/sec

7: The receivers reported horizontal speed after filtering must be less than 0.3 metres/sec.

Note: An unbroken pass on all selected checks for 10 seconds is required for the EKF2 to set its origin and start using GPS.

Note: The accuracy required for checks 2, 3, 5, 6 and 7 can be adjusted using the EK2\_CHECK\_SCALE parameter.

### **EK2\_CHECK\_SCALE**

This is a percentage scaler applied to the thresholds that are used to check GPS accuracy before it is used by the EKF. Values greater than 100 increase and values less than 100 reduce the maximum GPS error the EKF will accept. This modifies the checks enabled by bits 2, 3, 5, 6 and 7 in the EK2\_GPS\_CHECK parameter.

### **EK2\_IMU\_MASK**

This is a 1 byte bitmap controlling which IMUs will be used by EKF2. A separate instance of EKF2 will be started for each IMU selected.

- Set to 1 to use the first IMU only (default)
- Set to 2 to use the second IMU only
- Set to 3 to use the first and second IMU.

Additional IMU's up to a maximum of 6 can be used if memory and processing resources permit. There may be insufficient memory and processing resources to run multiple instances. If this occurs EKF2 will fail to start and the following message will be sent to the GCS console.

NavEKF2: not enough memory

If terrain data is not being used, some additional memory can be released by setting TERRAIN\_ENABLE=0 and rebooting.

## **ROS**

ArduPilot can be used with ROS using [Gazebo](#) and [MAVROS](#). The best tutorials are these [ErleRobotics Tutorials](#).

Instructions for using [Gazebo with ArduPilot](#) are [here](#) and has been [blogged about here](#).

VR Robotics has successfully used Rover with ROS for SLAM as demonstrated below.

We are keen to improve ArduPilot's support of ROS so if you find issues (such as commands that do not seem to be supported), please report them in the [ArduPilot issues list](#) with a title that includes "ROS" and we will attempt to resolve them as quickly as possible.

## **Pixhawk Advanced Hardware Info**

Basic information on the Pixhawk can be found in the [Supported Hardware Section](#).

Please follow the links below to learn more about potential Pixhawk boot issues and how to add a COA

(certificate of authenticity) to your Pixhawk boards.

## How to use the “auth” command to sign a Pixhawk Board with your Certificate of Authenticity

The essence of this process is an RSA private/public key pair and a signing process that uses these keys to put some unique information onto every PX4 board.

### Public/Private Key/s, SD Card, and Logging

The “auth” command will

- read a properly prepared public/private key data from the SD card
- use the key off the SD card to create the Certificate-Of-Authenticity (COA) in the One-Time\_programmable (OTP) ROM.
- log the results to a log file on the SD card, for your records. (optional)

### Bootloader

The bootloader must identify itself as revision 4 or later for this to work.

eg: px\_uploader.py should say something like: “Found board xxxx bootloader rev 4 on /xxxxxxxx”

### Firmware

The firmware must contain a directory Firmware/src/systemcmds/auth and have a recent “auth.c”.

- typically made and uploaded with “make px4fmu-v1\_auth upload”, or similar.
- verify you have a suitable version of the firmware loaded on your PX4 by connecting to the nsh in the usual way, and typing ‘auth’[enter]
- the “auth” command can do a bunch of stuff related to reading/writing/verifying/signing/logging of OTP data. It’s the main tool you’ll use ( see below). It can read/write public/private key data from SD card, or it can use a “hardcoded” TEST version.

### Preparing SD card (one time only)

- Using a tool on linux or OSX called ssh-keygen make a new “pair” of 1024bit RSA keyfiles like this:

```
ssh-keygen -t rsa -b 1024 -f rsa1024
```

- Generating public/private rsa key pair.
- Enter passphrase (empty for no passphrase): [just press enter]
- Enter same passphrase again: [just press enter, again]

Warning

#### This is

your PRIVATE key - do not share this file. back it up and keep it safe.

Tip

#### This is your

PUBLIC key - share it with all GCS makers, etc.

- Format the private key file to suit the PX4:

- Copy the rsa1024 to a new file, called “privatekey.txt”
- Edit the file with a text editor to remove the first and last lines of the file (they say `-----BEGIN RSA PRIVATE KEY-----` and `-----END RSA PRIVATE KEY-----`) and save it.
- Copy the **privatekey.txt** to a SD card which you will insert into a PX4 that is setup as per above. Do not use this method for the public key. See below.
- Make the public key file on the SD card:
  - With the SD card inserted, and the PX4 booted, use the *nsh shell*, and type:

```
auth -m
```

this will use the **privatekey.txt** on the SD card, and create a **publickey.txt** on the SD card (this file is needed for many of the auth commands to work).

## Validate it works

- There are lots of options to the “auth” command that you can use to test your configuration.
- Warning
- The only really \*hazardous\* options are `-w` and `-k`. Avoid these till you are sure everything else seems to be going well.
- It is a good idea to reboot each time you use the ‘auth’ command, as it’s very aggressive on RAM usage.
  - When you are sure you have everything OK, run `auth -k -l` (it will write a new COA to OTP, and lock it) and optionally `auth -v` to verify it worked.

## Automate running the ‘auth’ script from the SD card

- Make an “etc” folder on the SD card if one is not already there.
- Make a rc, or **rc.txt** file ( either works ) on the SD card, in the `/etc` folder if one is not already there.
- Edit the **rc.txt** file, and put the `auth -k -l` command in the file as you wish it to be run.
- (When booted with this card inserted it will make the PX4 flash the OTP area with the COA and log the results to **OTPCertificates.log**)

## Troubleshooting Pixhawk/PX4 Boot

This article explains how to check if the Pixhawk/PX4 has booted properly.

The following tests can help you determine if boot has failed, and possible causes:

- Check the Pixhawk/PX4 **LEDs** and **Sounds** as these can immediately confirm a successful boot. If boot fails, these can broadly indicate the point of failure.
- Check that the board has appropriate ArduPilot firmware installed.
- Ensure the memory card is fully inserted into the Pixhawk/controller card socket.
- **Check the boot log:**
  1. Remove the SD card from the board and insert into your regular computer
  2. open the **APM/boot.log** file and check that it contains the following lines

```
Starting APM sensors
Trying PX4IO board
PX4IO board OK
```

```
Starting ArduPilot
rc.APM finished
```

3. If the file does not exist or the contents are different from above then reinstall the Copter/Plane/Rover firmware.

## MAVProxy Developer GCS

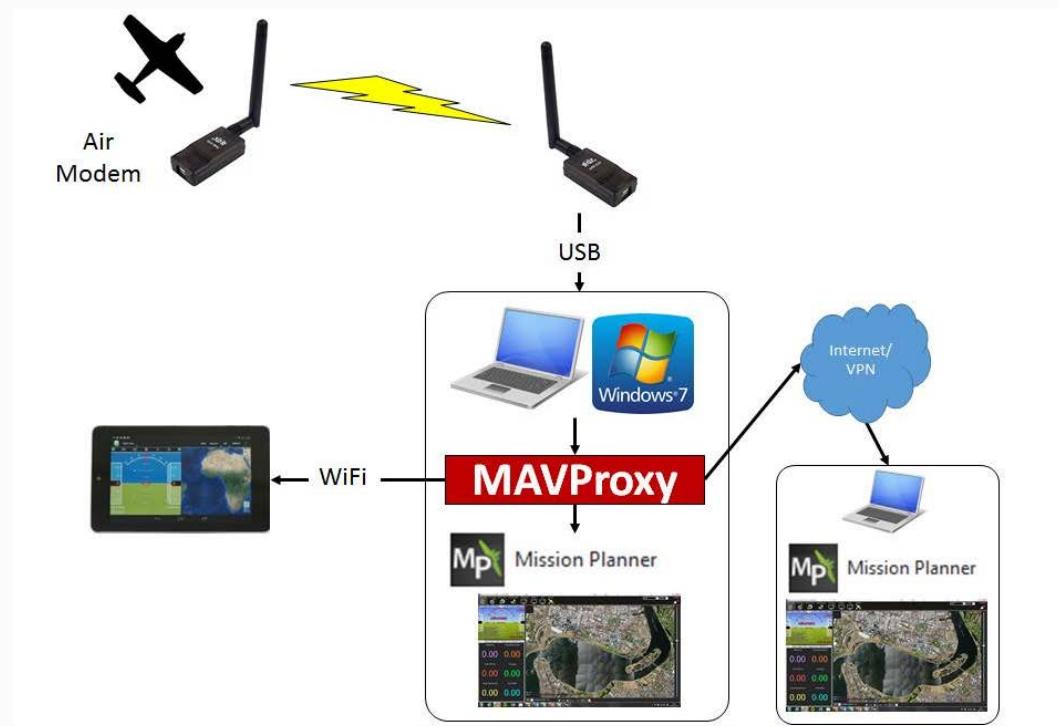
[MAVProxy](#) is a powerful command-line based “developer” ground station software that complements your favorite GUI ground station, such as *Mission Planner*, *APM Planner 2* etc. It has a number of key features, including the ability to forward the messages from your UAV over the network via UDP to multiple other ground station software on other devices.

MAVProxy is commonly used by developers (especially with SITL) [for testing new builds](#).

The [official MAVProxy documentation](#) contains detailed information on how to use the tool. This section contains complementary articles and tutorials:

### MAVProxy on Windows 7

This guide shows how to set up MAVProxy to allow forwarding of messages via network interfaces.



### Overview

[MAVProxy](#) is a powerful command-line based “developer” ground station software that complements your favorite GUI ground station, such as *Mission Planner*, *APM Planner 2* etc.

A key feature of MAVProxy is its ability to forward the messages from your UAV over the network via UDP to multiple other ground station software on other devices. For example: you can run a ground station on a laptop next to your antenna and forward via wifi to a smartphone/tablet which lets you easily relocate to launch into wind before heading back to your fixed antenna. I have also used it to send telemetry data to a friend acting as spotter several kilometers away (via 4G vpn) during a longer flight so that he could

monitor the entire flight and determine where to look to find the aircraft in flight.

This guide shows how to set up MAVProxy to allow forwarding via network interfaces and usage via command line. There may be other ways to get this running and you may need other packages as per the [official MAVProxy documentation](#) in order to use more advanced functions. No warranty responsibility for damage etc.

#### Note

Full credit to Andrew Tridgell and all other contributors to MAVProxy and the other software used here.

### **Step 1: Check you can connect to your UAV**

Before starting anything make sure you can make a direct connection to your aircraft with your normal Ground Station software on the PC in question. Check that you know the correct COM port and baudrate for the modem attached to your laptop as we will need that info later.

### **Step 2: Install MAVProxy**

Install MAVProxy from <http://firmware.ardupilot.org/Tools/MAVProxy>

### **Step 3: Ready to run**

Check your radio modem is connected via USB, the aircraft modem and APM are powered and ensure any other ground station software is closed.

Open a command prompt window (Click start, type cmd and then press enter) and then run:

```
"C:\Program Files (x86)\MAVProxy\mavproxy.exe"
```

If everything has worked you should see MAVProxy start up and some basic flight data such as mode and current waypoint appear. Occasionally some data does seem to result in glitching and odd characters appearing onscreen but this does not seem to affect reliability or performance.

```
C:\Windows\system32\cmd.exe - mavproxy.py --master="com3" --baudrate 57600
C:\Python27\MAVProxy-1.3.3\MAVProxy>mavproxy.py --master="com3" --baudrate 57600
Logging to mav.tlog
MAV> MANUAL> Mode MANUAL
waypoint 65535
GPS lock at 73 meters
```

Enter a command such as mode FBWA and press enter. You should see MAVLink report the mode change and notice your aircraft change behaviour into that mode.

The full list of MAVLink commands can be found at <http://ardupilot.github.io/MAVProxy> if you want to experiment further with the command line.

To exit MAVLink press Control+C together.

## Step 4: Forwarding over network

To forward the MAV data over the network including to a local program on your PC we simply need to add some extra parameters when starting MAVProxy via the command line.

To connect with a local ground station software such as Mission Planner start MAVProxy as above with the command `mavproxy.exe -master="com14" -baudrate 57600 -out 127.0.0.1:14550` and press enter.

Then open Mission Planner and select UDP and click connect. Clink OK on the default prompt for port number (14550) and you should see mission planner start downloading parameters and connecting to your UAV.

Finally you can add the IP address of any computer to forward the telemetry stream onwards to other ground stations.

1. On the local network/wifi you will need to ensure there is no firewall on the client PC stopping the incoming stream to your ground station software.
2. Add `-out IP_ADDRESS:14550` to the end of the `mavproxy.exe` command. You can add as many separate `-out` parameters as you want depending on how many extra ground stations you are running.
3. Set each ground station to listen for UDP packets on port 14550

## Plane SITL/MAVProxy Tutorial

This tutorial provides a basic walk-through of how to use [SITL](#) and [MAVProxy](#) for *Plane* testing.

### Overview

The article is intended primarily for developers who want to test new Plane builds and bug fixes using SITL and MAVProxy. It shows how to take off, run missions, fly in [GUIDED](#) mode, set a geofence, and perform a number of other basic testing tasks.

The tutorial is complementary to the topic [Using SITL for ArduPilot Testing](#).

#### Note

- We use MAVProxy here, but you can [attach another ground station to SITL](#) if you prefer (similar instructions can be used in any GCS).
- This tutorial is for Plane - see [Copter](#) and [Rover](#) for similar tutorials on the other vehicles.

### Preconditions

The tutorial assumes you have already set up [SITL on Windows](#) or [Linux](#) and that you have started SITL using the `--map` and `--console` options:

```
cd ~/ardupilot/ArduPlane
sim_vehicle.py -j4 --map --console
```

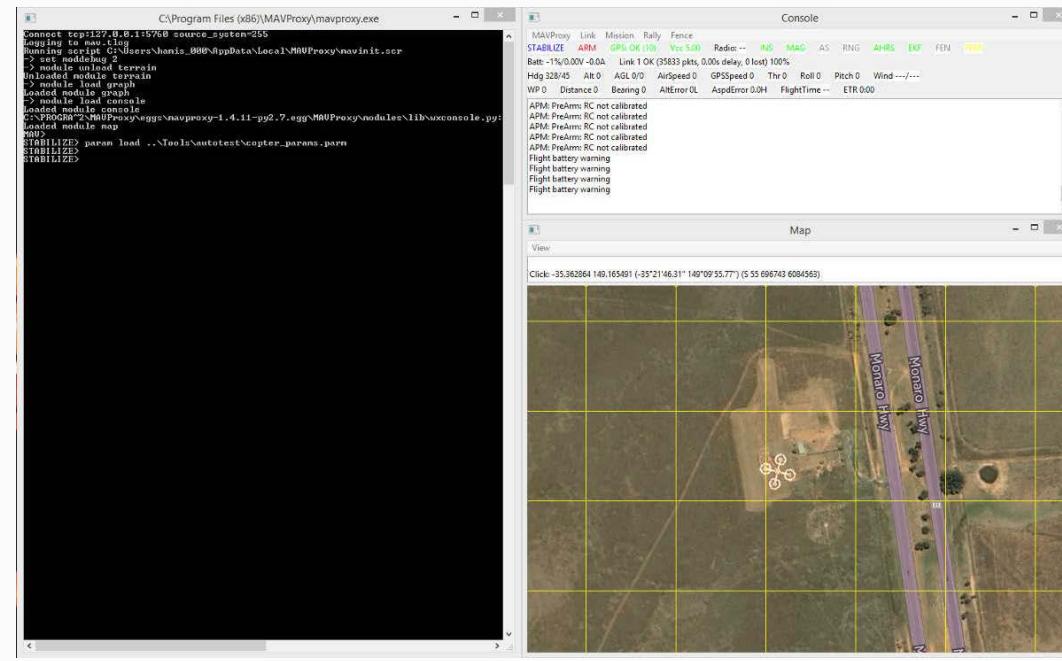
As part of the setup you should have loaded some standard/test parameters into the *MAVProxy Command Prompt*:

```
param load ..\Tools\autotest\default_params\plane-jsbsim.prm
```

#### Note

- Some arduplane versions' param path is not same as this, if fail to open file, you should check param path.
- In ardulink/ArduPlane file, there is a mav.parm with all parameters. If you want to change more params you can edit it and load it.

The *MAVProxy Command Prompt*, *Console* and *Map* should be arranged conveniently so you can observe the status and send commands at the same time.



## Taking off

To take off with Plane you should use a mission (AUTO mode) that contains the [MAV\\_CMD\\_NAV\\_TAKEOFF](#) command. Once you are airborne you can switch to other other [flight modes](#).

### Note

At time of writing, Plane only supports automated takeoff in [AUTO](#) mode as part of a mission. If you want to change to [GUIDED](#) mode (or any other mode) then you first have to take off using a mission.

First load the **CMAC-circuit.txt** test mission using the [wp load](#) command as shown below (this mission contains the takeoff command):

```
wp load ..\Tools\autotest\CMAC-circuit.txt
wp list
```

Once the mission is loaded you can take off by arming the throttle and setting the mode to [AUTO](#) (the order of these operations does not matter):

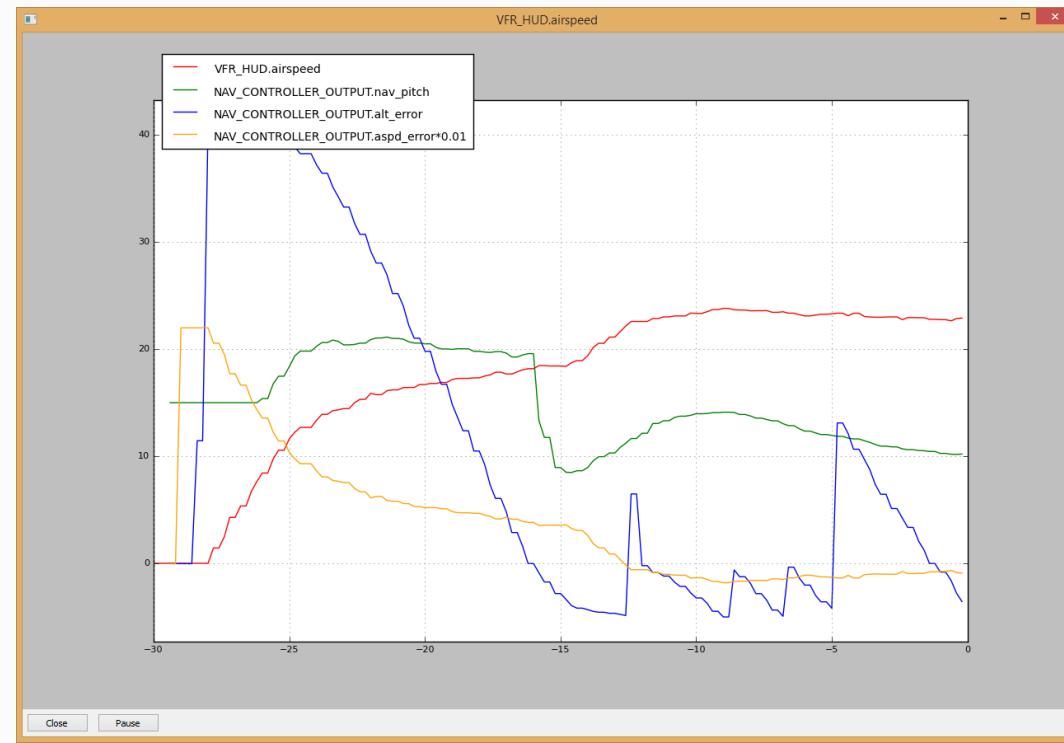
```
mode auto
arm throttle
```

The plane should take off to an altitude of 40 metres and then proceed through the other mission waypoints (most of which have an altitude of 100m). You can stop/pause the mission at any time by changing the mode.

## Monitoring

During takeoff you can watch the altitude increase on the console in the *Alt* field.

Developers may find it useful to **graph** the takeoff by first entering the `gtakeoff` command.



## MAVProxy: PlaneTakeoff Graph (gtakeoff)

### Troubleshooting

The most common sources of difficulty taking off are:

1. Using a mission that does not contain a takeoff command!
2. Attempting to takeoff when the vehicle is not armed. This can happen if the vehicle fails pre-arm checks.

You can list all *enabled* checks using the command `arm list`:

```
LAND> arm list
LAND> all
params
voltage
compass
battery
ins
rc
baro
gps
```

You can enable and disable checks using `arm check n` and `arm uncheck n` respectively, where n is the name of the check. Use `n` value of `all` to enables/disable all checks.

### Flying a mission

You can load a mission at any time using the `wp load` command. The mission will start as soon as the vehicle is armed and you're in `AUTO` mode.

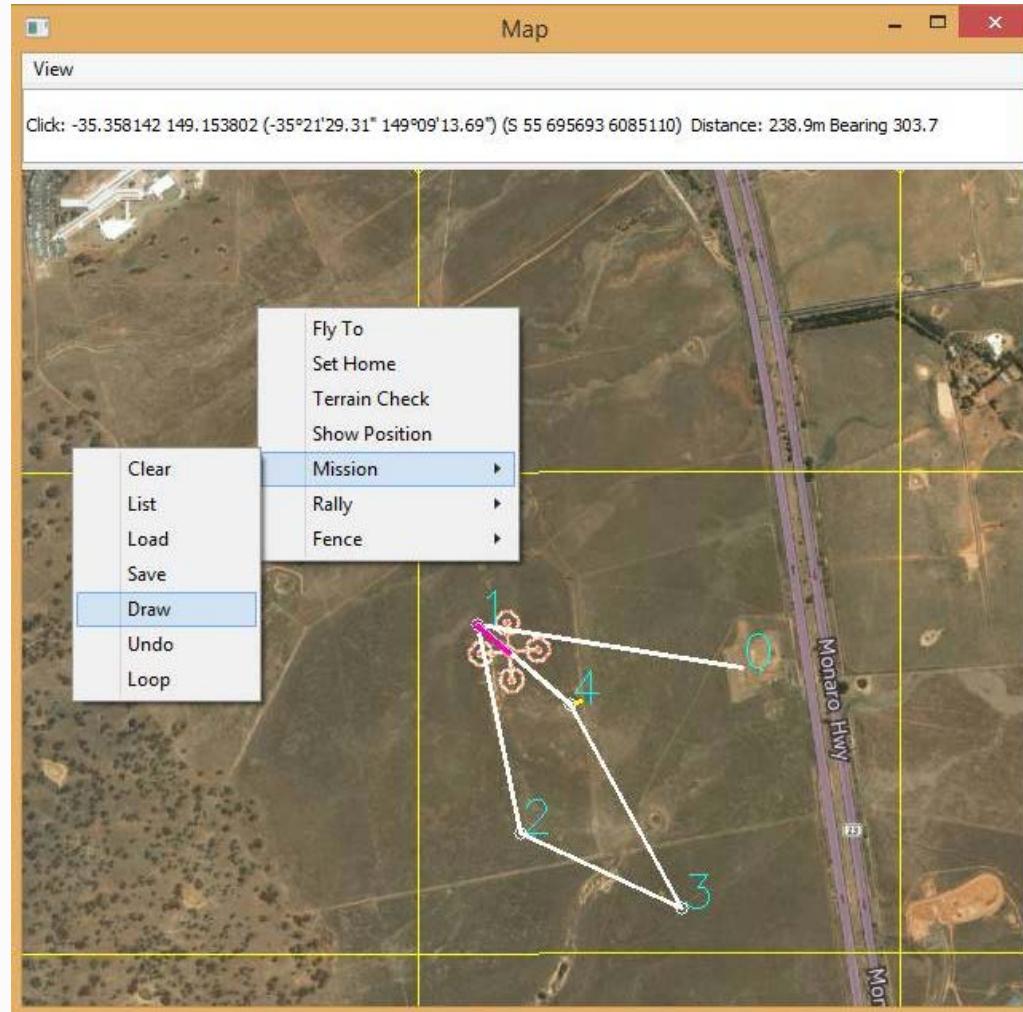
The example below shows how to load and start one of the test missions, skip to the second waypoint using `wp set n`, and *loop* the mission:

```
wp load ..\Tools\autotest\CMAC-circuit.txt
mode auto
wp set 2
wp loop
```

The [MAVProxy Waypoints documentation](#) lists the full set of available commands (or you can get them using auto-completion by typing "wp" on the command line).

If you want to create a waypoint mission, this is most easily done on the map:

1. Right-click on the map and then select **Mission | Draw**.



### **MAVProxy: Draw Mission Menu**

2. Left-click on the map where you want the points to appear.

#### Note

Nothing visible will happen when you make the first click. After the second click, lines will join your points to show the path.

3. When you're done, you can loop the mission by right-clicking on the map and selecting **Mission | Loop**.

This approach only allows you to create `MAV_CMD_NAV_WAYPOINT` commands. You can edit missions and use other commands on Linux using the `misseditor` module (`module load misseditor`). This is currently broken on Windows. It is also possible to load other types of commands from files.

## Changing flight modes

Plane supports a [number of flight modes](#), which you can list in *MAVProxy* using the `mode` command:

```
AUTO> mode
(AUTO> ''LAND', 'AUTOTUNE', 'STABILIZE', 'AUTO', 'GUIDED', 'LOITER', 'MANUAL', 'FBWA', 'FBWB', 'CRUISE',
'INITIALISING', 'CIRCLE', 'ACRO'')
```

You can set the mode by entering `mode MODENAME` on the *MAVProxy command prompt*.

For example, the command below shows how to put Plane into [CIRCLE mode](#) (this is like Loiter, except that the plane does not attempt to hold position).

```
mode circle
```

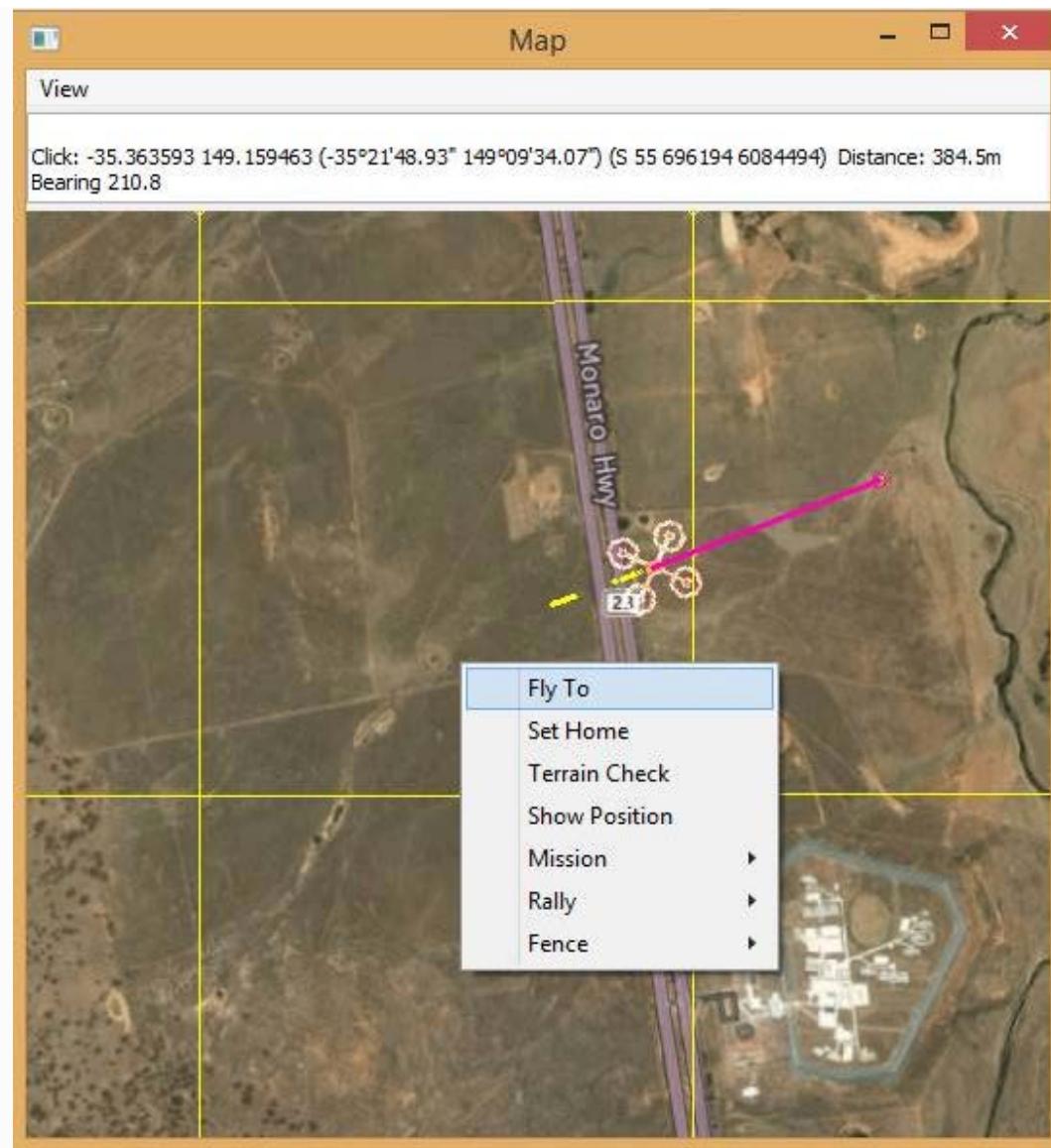
## Guiding the vehicle

Once you've taken off you can move the vehicle around the map in [GUIDED](#) mode. Plane will fly to a specified point, and then circle it.

First change the mode:

```
mode guided
```

The easiest way to set a target point is to right-click on the map where you want to go, select **Fly to**, and then enter the target altitude.



## MAVProxy: Fly toLocation

You can also enter the target position manually on the command line using the two formats below. If only the altitude is specified, the last specified LAT/LON will be used.

```
guided ALTITUDE
guided LAT LON ALTITUDE
```

### Note

Unlike with Copter, you can't do much with Plane in `GUIDED` mode because the mode does not support many commands. This mode is primarily useful for simply flying to a point.

### Setting a GeoFence

A GeoFence is a virtual barrier that Plane uses to constrain the movement of the vehicle (and move it to a safe location if control is lost). Plane allows you to specify an arbitrarily shaped region on the map for the fence, and an upper and lower altitude. If the fence is breached, Plane will fly to the centre of the fence and circle (or a rally point). [Geo-Fencing in Plane](#) describes the fence in more detail.

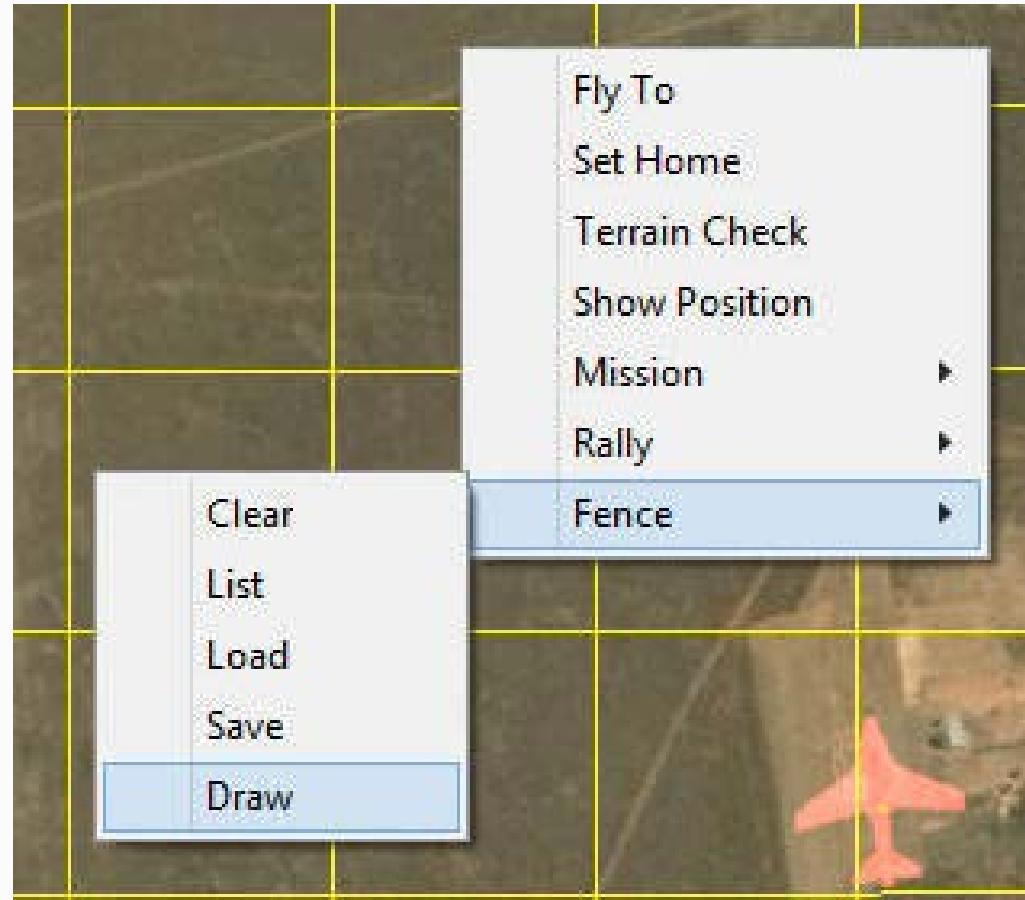
The fence behaviour is set using the [Plane Fence Parameters](#). You can list these with `param show`:

```
GUIDED> param show fence*
```

```
GUIDED> FENCE_ACTION      1.000000
FENCE_AUTOENABLE 0.000000
FENCE_CHANNEL     0.000000
FENCE_MAXALT      0.000000
FENCE_MINALT      0.000000
FENCE_RETALT      0.000000
FENCE_RET_RALLY   0.000000
FENCE_TOTAL       7.000000
```

Creating the fence is very similar to creating a waypoint mission:

1. Right-click on the map and then select **Fence | Draw**.



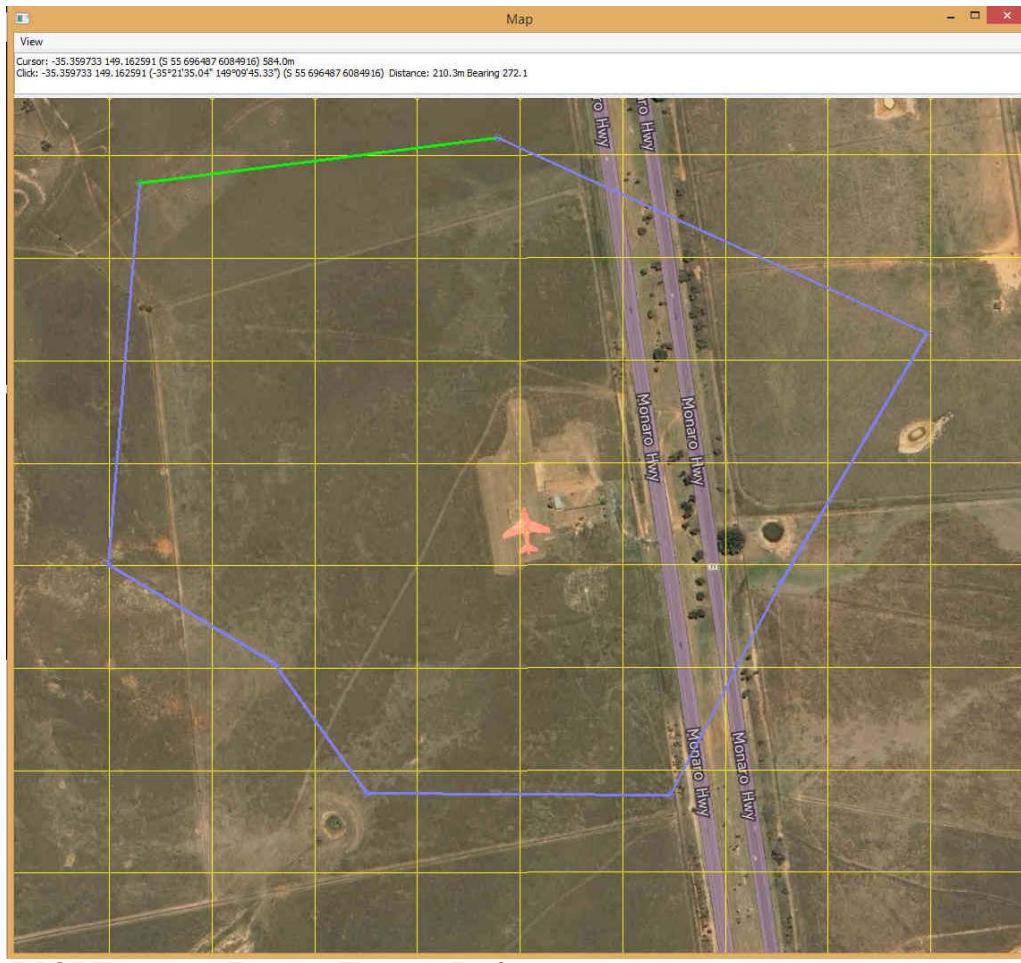
**MAVProxy: Draw Fence Menu**

2. Left-click on the map at points where you want the fence “posts” to appear.

#### Note

Nothing visible will happen when you make the first click. After the second click, lines will join your points to show the path.

3. When you're done, you can loop the fence by right-clicking on the map.



### **MAVProxy: Draw Fence Points**

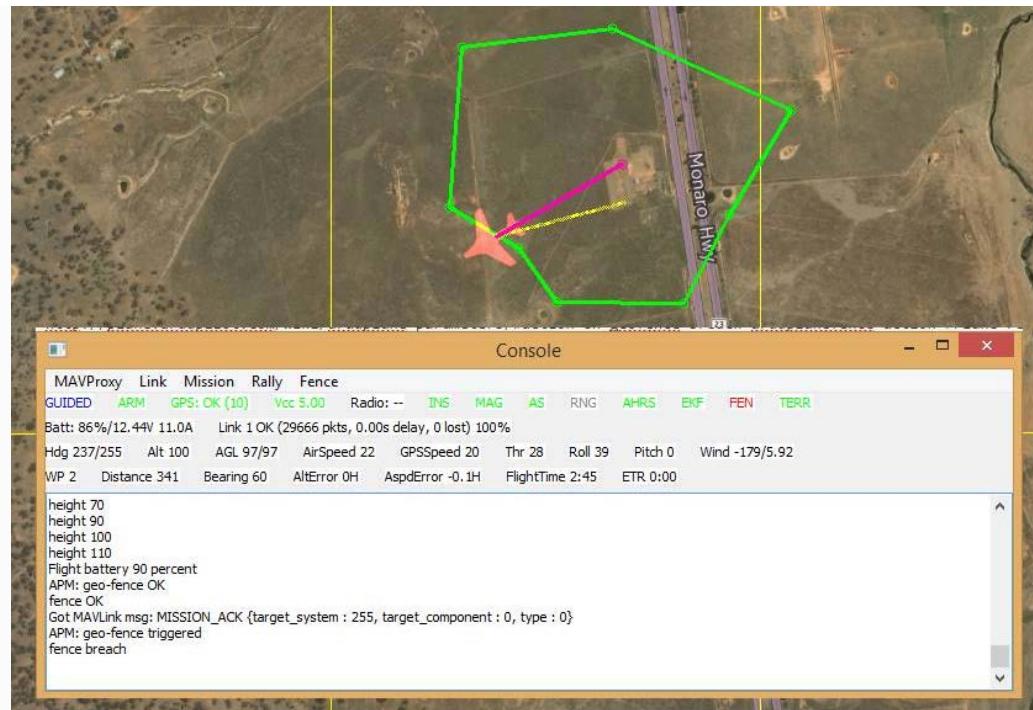
4. The fence is initially disabled. To turn it on set the value to one:

```
GUIDED> fence enable
```

5. Now lets make the plane cross the barrier. Assuming you are already flying you can use the following to make it fly straight ahead into the fence:

```
GUIDED> mode cruise
```

6. When the fence is crossed, the plane will fly to the centre of the fence region and then circle. The console shows that the breach has occurred.



## MAVProxy: Fence Breach shown on Console and Map

Instead of flying to the centre of the fence you can instead add a [rally point](#) to the map and fly to it by enabling the parameter [FENCE\\_RET\\_RALLY](#).

## Testing the vehicle

MAVProxy allows you to list all the parameters affecting the vehicle and simulation using [param show \\*](#), and to set any parameter using: [param set PARAM\\_NAME VALUE](#). In addition to affecting the vehicle itself some parameters simulate the performance/failure of specific hardware components and the environment (for example, the wind). These can be listed using: [:ref:`param show sim`](#). The topic [Using SITL for ArduPilot Testing <using-sitl-for-ardupilot-testing>](#) explains more about how you can test using SITL.

## Rover SITL/MAVProxy Tutorial

This tutorial provides a basic walk-through of how to use [SITL](#) and [MAVProxy](#) for *Rover* testing.

### Overview

*Rover* is an easy platform to get started with. Unlike Copter or Plane there are no fast-moving propellers, so there is no need to arm motors. As movement is along the ground, there is no need to consider taking off or landing, and pausing/loitering/waiting is just “stopping”. At the end of initialization the *Rover* is ready to go!

The article is intended primarily for developers who want to test new *Rover* builds and bug fixes using [SITL](#) and [MAVProxy](#). It shows how to use the different modes, run missions, set a geofence, and perform a number of other basic testing tasks.

The tutorial is complementary to the topic [Using SITL for ArduPilot Testing](#).

### Note

- We use [MAVProxy](#) here, but you can [attach another ground station to SITL](#) if you prefer (similar instructions can be used in any GCS).

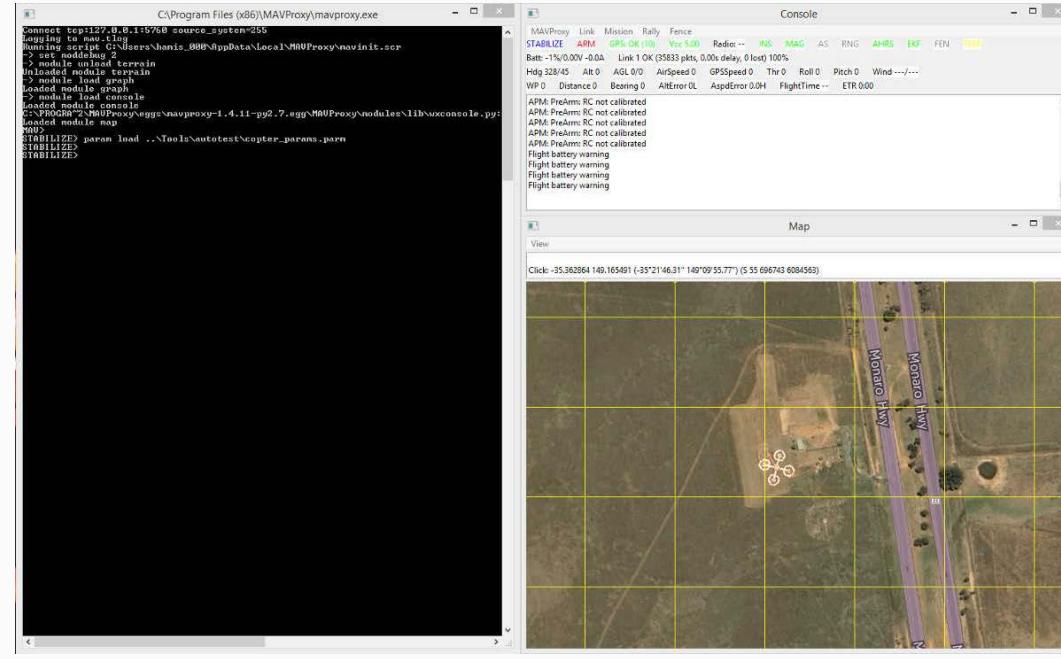
- This tutorial is for Rover - see [Copter](#) and [Plane](#) for similar tutorials on the other vehicles.

## Preconditions

The tutorial assumes you have already set up [SITL](#) on [Windows](#) or [Linux](#) and that you have started SITL using the `--map` and `--console` options:

```
cd ~/ardupilot/APMrover2
sim_vehicle.py -j4 --map --console
```

Arrange the *MAVProxy Command Prompt*, *Console* and *Map* conveniently so you can observe the status and send commands at the same time.



### Note

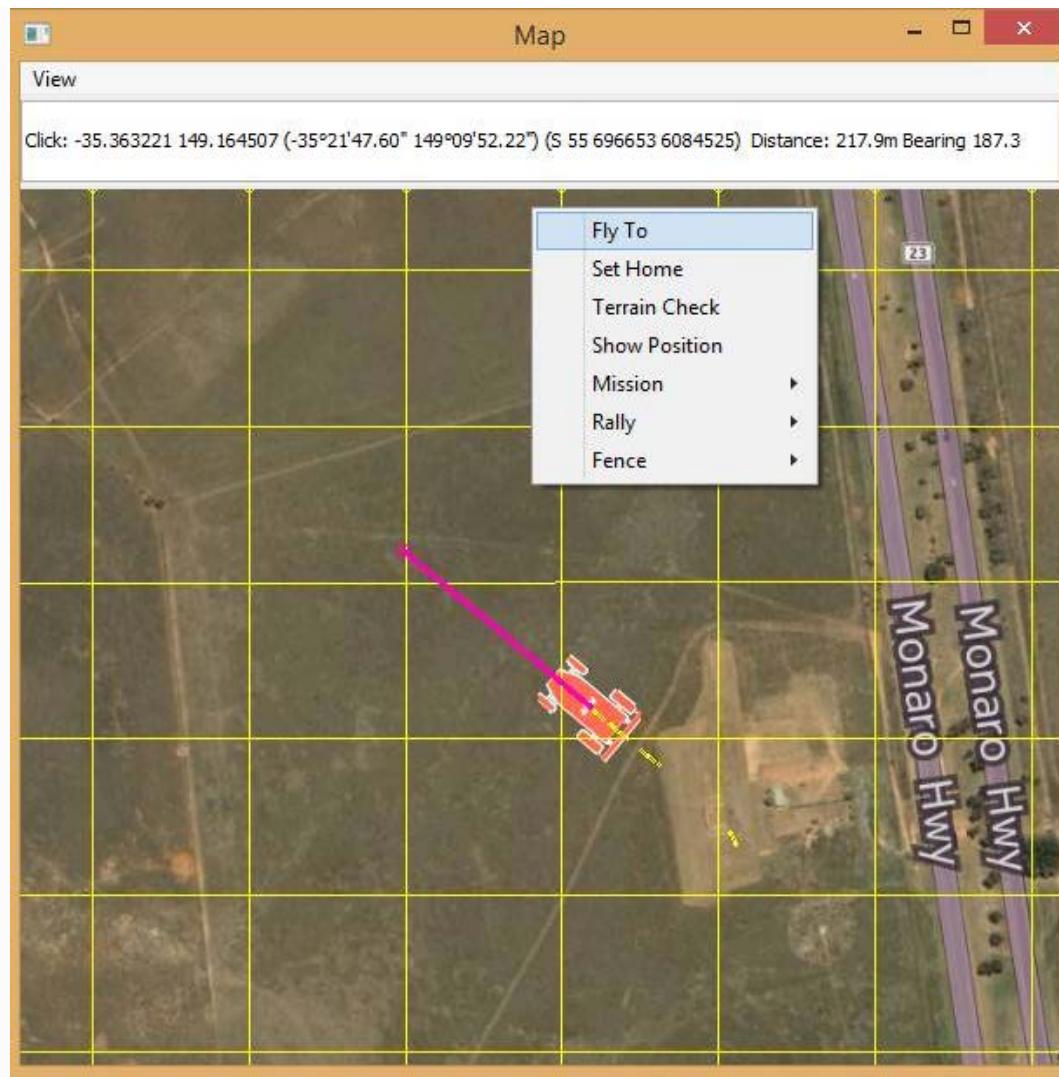
As part of the setup you may have loaded some [standard/test parameters](#) into the *MAVProxy Command Prompt*:

```
param load ..\Tools\autotest\Rover.prm
```

Unlike for Copter and Plane, this is not strictly necessary. Rover is a forgiving platform!

## Starting

**Right-click** on the map at your target destination, select **Fly to**, and enter an altitude (which is ignored). Rover will change to **GUIDED** mode, drive to the location, and then pause (still in GUIDED mode).



## **MAVProxy: Rover go to location**

If you have a mission loaded, you can also simply change to AUTO mode to run it:

```
mode auto
```

### Tip

Any commands or parameters that are not relevant to a Rover are ignored (e.g. TAKEOFF command and any altitude information).

### **Guiding the vehicle**

The previous section covered almost everything you need to know about moving the vehicle around the map in **GUIDED** mode.

You can also enter the target position manually on the command line using the two formats below. If only the altitude is specified, the last specified LAT/LON will be used.

```
guided ALTITUDE
guided LAT LON ALTITUDE
```

Just specifying the altitude (as 0) is useful if you need to pause then restart a guided path. The commands below set the mode to HOLD and then restart the vehicle travelling towards the previous point:

```
GUIDED> mode hold
HOLD> guided 0
GUIDED>
```

When you're finished you can return to the initial position by changing to `RTL` mode (this also works in `AUTO` mode):

```
GUIDED> mode rtl
RLT>
```

## Running a mission

You can load a mission at any time using the `wp load` command. After you've taken off the current mission will start as soon as you change to `AUTO` mode.

The example below shows how to load and start one of the test missions, skip to the second waypoint, and *loop* the mission:

```
wp load ..\Tools\autotest\CMAC-circuit.txt
mode auto
wp set 2
wp loop
```

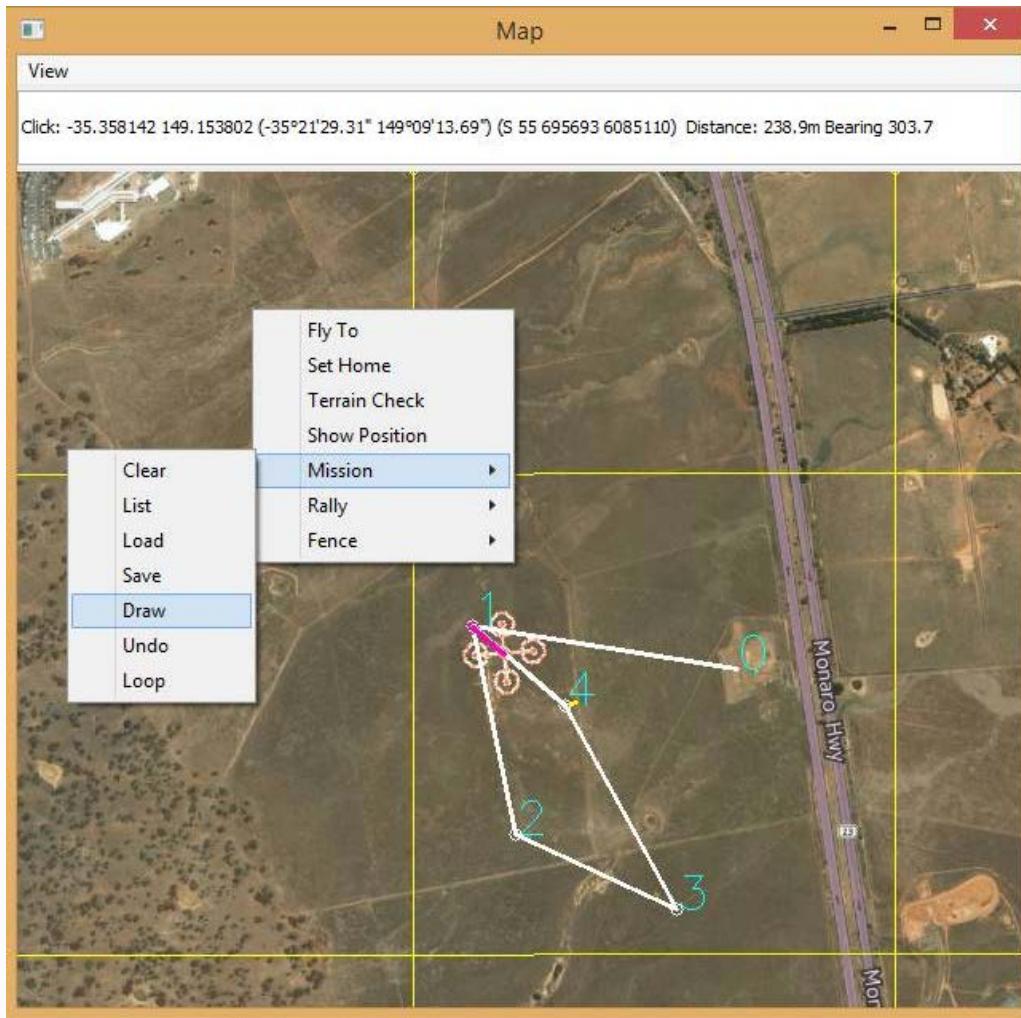
### Note

Rover will only run [commands it supports](#) (others are ignored).

The [MAVProxy Waypoints documentation](#) lists the full set of available commands (or you can get them using auto-completion by typing “wp” on the command line).

If you want to create a waypoint mission, this is most easily done on the map:

1. Right-click on the map and then select **Mission | Draw**.



**MAVProxy: Draw Mission Menu (This image is for Copter, but the behaviour is the same).**

2. Left-click on the map where you want the points to appear.

Note

Nothing visible will happen when you make the first click. After the second click, lines will join your points to show the path

3. When you're done, you can loop the mission by right-clicking on the map and selecting **Mission | Loop**.

This approach only allows you to create `MAV_CMD_NAV_WAYPOINT` commands. You can edit missions and use other commands on Linux using the `misseditor` module (`module load misseditor`). This is currently broken on Windows. It is also possible to load other types of commands from files.

Tip

At any point you can pause the mission by setting the mode to `HOLD`, and restart by setting it back to `AUTO`.

### Changing mode

Rover supports a small number of modes which you can list in MAVProxy with the `mode` command:

```
GUIDED> mode
GUIDED> ('Available modes: ', ['AUTO', 'GUIDED', 'MANUAL', 'LEARNING', 'RTL', 'INITIALISING', 'HOLD',
'STEERING'])
```

The useful modes for simulation are:

- `AUTO` - Run a mission
- `GUIDED` - Move where directed by GCS
- `RTL` - Return to launch
- `HOLD` - WAIT - pause mission/stop moving.

As shown in the previous section, you can change the mode by specifying `mode modename`. Most of the modes can be set by just entering the mode name, e.g. `rtl`, `auto`, `hold`.

## Testing the vehicle

MAVProxy allows you to list all the parameters affecting the vehicle and simulation using `param show *`, and to set any parameter using: `param set PARAM_NAME VALUE`. In addition to affecting the vehicle itself some parameters simulate the performance/failure of specific hardware components and the environment (for example, the wind). These can be listed using: `:ref:`param show sim``. The topic *Using SITL for ArduPilot Testing <using-sitl-for-ardupilot-testing>* explains more about how you can test using SITL.

## Ready-to-Fly (RTF) Vehicle Developer Information

This section is for developer information related to “non-standard” RTF vehicles based on ArduPilot.

### Note

ArduPilot-based UAVs are often very similar from a developer perspective; developers upload standard firmware direct to the flight controller using Mission Planner (or other GCS software). This section is for vehicles that use a non-standard approach, including modified source, different approaches for uploading firmware, support for companion computers, etc.

## 3DR Solo (Developer Information)

This page provides instructions for developers working with the 3DR Solo Ready-to-fly copter. The comprehensive [3DR solo dev guide can be found here](#).



## Uploading custom firmware to the Pixhawk2

- Solo can be flown with ardupilot/master but when shipped by 3DR it runs a modified version of APM:Copter3.3.
  - The firmware can be found on Github here.
  - The source code can be found here: [ardupilot](#), [PX4Firmware](#), [NuttX](#).
- Compile instructions are the same as for master ardupilot for a Pixhawk with the exception that the modified repos should be used.
- After modifying/building the firmware, turn on the Solo and connect your PC's network to the Solo controller's access point using the same login/password used by the Solo app (by default these are something like: [SoloLink\\_XXXXXX/sololink](#)).
- Use scp to copy the ArduCopter-v2.px4 file from your PC to **root@10.1.1.10:/firmware** (i.e. the firmware directory on the IMX6 on the Solo vehicle). Windows users in particular may want to install [Putty](#) and use a command like below:

```
C:\Program~2\PuTTY\pscp.exe -pw SSH-PASSWORD -scp ArduCopter-v2.px4 root@10.1.1.10:/firmware/
```

### Note

The SSH-PASSWORD should be replaced with the actual ssh password for your vehicle. This password has not yet been officially released by 3DR, but we expect it will be in the near future.

- Reboot the vehicle and the vehicle's IMX6 should load the firmware onto the Pixhawk2 (you should see the regular rainbow colours on the arm LEDs as the upload proceeds).
- After uploading the firmware is moved to the **/firmware/loaded** directory.

## Changing the controller mode

- ssh into the iMX6 board in your artoo (ssh **root@10.1.1.1**)

```
vi /firmware/cfg/stick-cfg-evt.cfg:
```

**Mode 1:**

```
throttle stick-id = 1, dir = 0
roll stick-id = 2, dir = 0
pitch stick-id = 3, dir = 1
yaw stick-id = 0, dir = 0
```

**Mode 3:**

```
throttle stick-id = 1, dir = 0
roll stick-id = 0, dir = 0
pitch stick-id = 3, dir = 1
yaw stick-id = 2, dir = 0
```

**Mode 4:**

```
throttle stick-id = 3, dir = 0
roll stick-id = 0, dir = 0
pitch stick-id = 1, dir = 1
yaw stick-id = 2, dir = 0
```

- Copy **/usr/bin/runStickCal.sh** to a new file (in the same directory) called **runStickMapper.sh**
- Edit **runStickMapper.sh** and replace:

```
stick-cal.py /dev/ttymxc1
```

with

```
stick-axis-cfg.py /dev/ttymxc1 /firmware/cfg/stick-cfg-evt.cfg
```

**User supplied teardown video**

© Copyright 2016, ArduPilot Dev Team.

