QGC: What you don’t know

Inside the open source ground control station software

Andrew Wilkins

Why am I giving this talk?

Ascend
Engineering

QGC: If you didn’t know

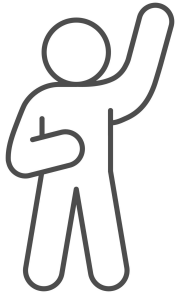
QGroundControl (QGC) is an open source ground control station for UAVs

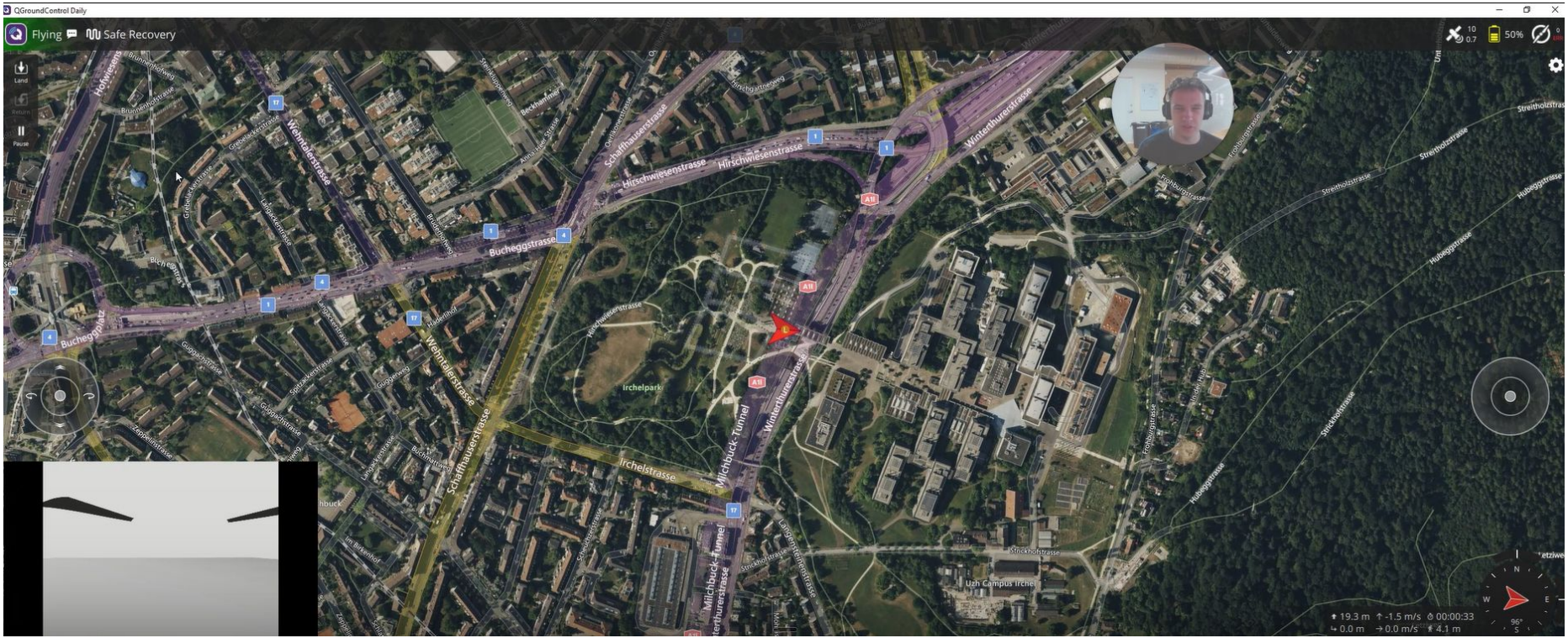
Built on the **Qt framework**, QGC runs natively on Windows, macOS, Linux, iOS, and Android from a single codebase, making cross-platform support a feature rather than a afterthought.

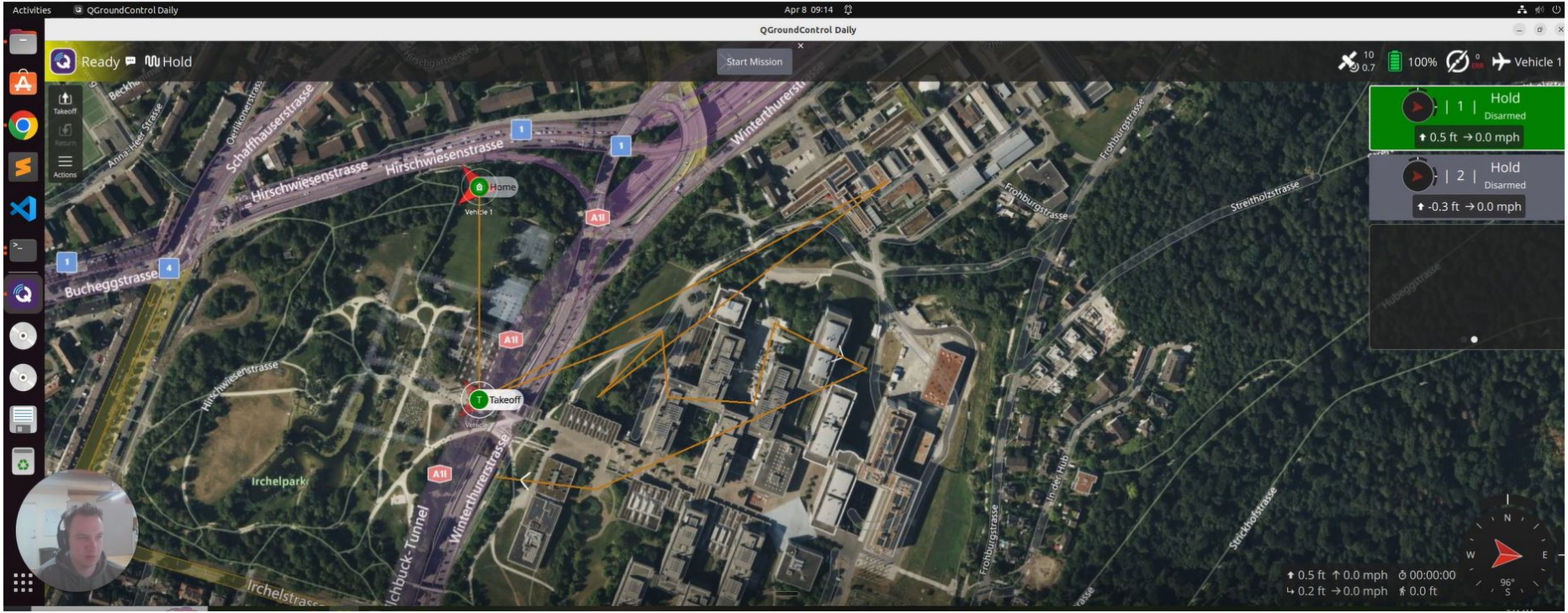
QGC utilizes **MAVLink**, the lightweight messaging protocol for drone communication, and supports both **ArduPilot** and **PX4** autopilot stacks.

The robust support QGC offers is why it's become the default ground control station across a huge range of commercial UAV companies, research institutions, and hobbyist communities.

The project has been driven primarily by **Donald Gagne**, who started building it as a personal project after retiring. QGC has since turned into one of the most widely deployed pieces of open source UAV software.




QGroundControl Daily
Flying
Safe Recovery
Land
Return
Pause
50%
Hirschwiesenstrasse
Winterthurerstrasse
Bucheggstrasse
Bucheggplatz
Wehntalerstrasse
Schaffhauserstrasse
Irchelpark
Irchelstrasse
Milchbuck-Tunnel
Uzh Campus Irchel
Streitholzstrasse
Hubeggstrasse
Strickhofstrasse
19.3 m
-1.5 m/s
00:00:33
0.0 m
0.0 m/s
4.1 m


Activities
QGroundControl Daily
Apr 8 09:14
QGroundControl Daily
Ready
Hold
Start Mission
Takeoff
Return
Actions
100%
Vehicle 1
Hold
Disarmed
0.5 ft
0.0 mph
Hold
Disarmed
-0.3 ft
0.0 mph
Home
Vehicle 1
Takeoff
Vehicle 1
Hirschwiesenstrasse
Bucheggstrasse
Irchelpark
Winterthurerstrasse
Frohburgstrasse
Streitholzstrasse
Standhofstrasse
A11
0.5 ft
0.0 mph
00:00:00
0.2 ft
0.0 mph
0.0 ft


Disconnected
A
B
C
D

Overview

New, Lesser-known and Hidden Features/Functionality in QGC

- Console Logging Categories
- Android SDL3 Migration
- Bluetooth LE
- Libarchive
- Unit Testing & Code Coverage
- Lua Scripting Tab
- Advanced vs Simple Mode
- Soon: Rebuilding the Video Pipeline

Console Logging (category hierarchy)

The go-to Debugging Weapon

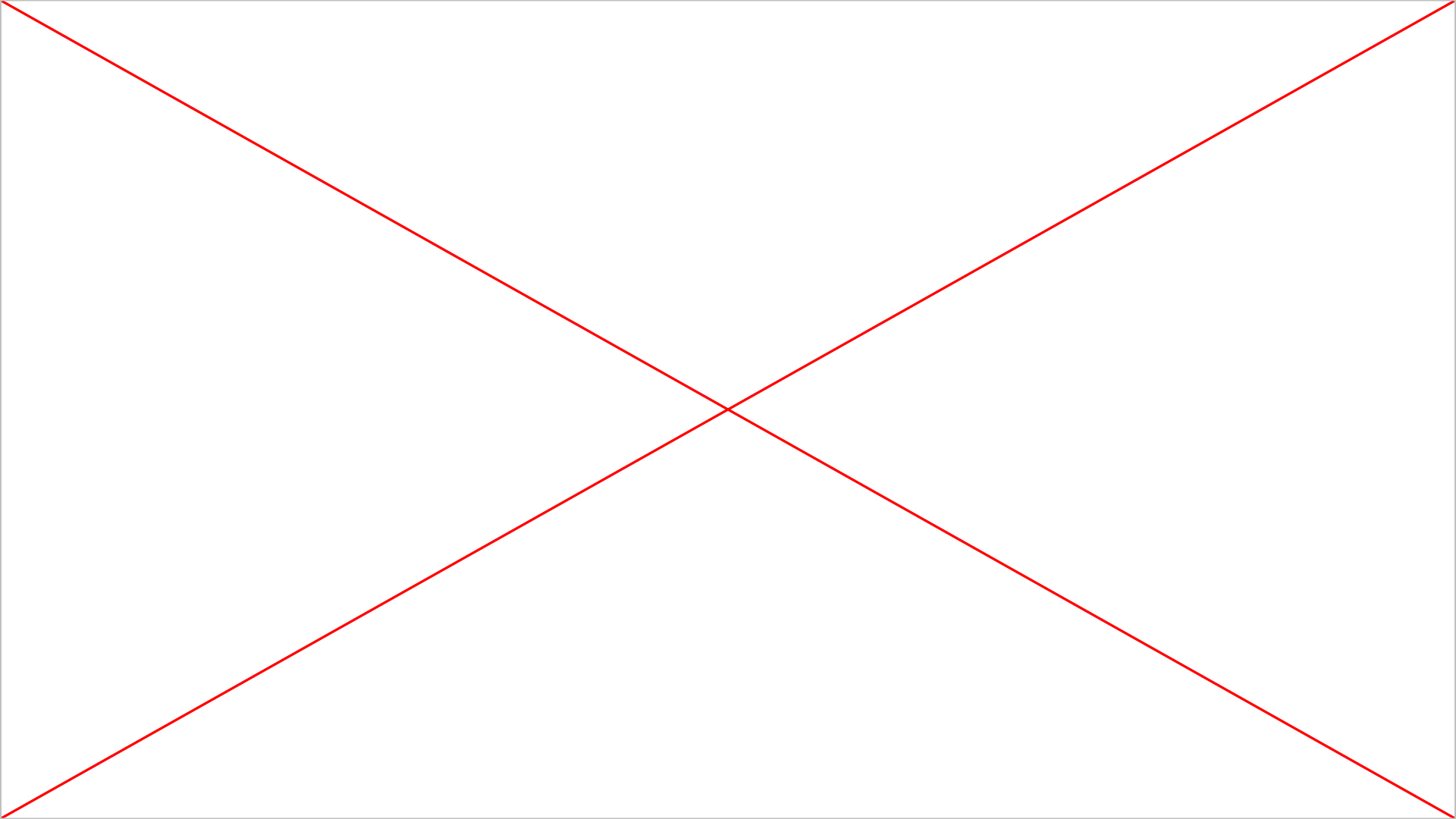
- When something goes wrong in the field or testing in development, the live console build into right into the app is a great place to start.
- Customizable console allows for streamlined debugging

Where to find it: Application Settings → 💬Logging

Implemented by Holden Ramsey

What you can do

- Select different logging categories e.g.
  - LinkManager (Comms)
  - ParameterManager (FactSystem)
  - FirmwareUpgrade (FirmwarePluggin)
  - PlanManager/MissionManger
- Hierarchical category selection
  - Select Debug levels for each category/subcategories
    - Debug
    - Info
    - Warning
    - Critical



Android Joystick SDL3 Refactor

Previous Problem:

QGC used SDL2 for joystick & game controller support on desktop (Windows, Linux, MacOS), however android needed a completely separate Java native interface workaround. Under the hood, two separate codebases handled the same job contingent on the operating system.

The Solution:

A full refactor overhaul by Holden Ramsey migrating both codebase paths to a single SDL3 based backend that behaves identically across operating systems.
On android, the deprecated JoystrickAndroid class was replaced with its own Java layer. Inputs are now forwarded from QGCActivity into SDL.

New Capabilities thanks to SDL3

- Gyroscope and accelerometer sensor input
- Touchpad and trackball support
- RGB LED control and player indicator LEDs
- Rumble and trigger haptics
- Battery level monitoring on the controller
- Type-specific button labels (knows if you're on Xbox vs. PlayStation layout)

Allows for end user to have a joystick indicator in the toolbar to show controller battery level and connection status.

Further SDL improvements/implementations are pending.

What does this mean?

Bluetooth LE

Previous Problem:

QGC supports Bluetooth Classic, but had no support for BLE. User wanting to use BLE MAVLink devices would get “connection failed” errors with no easy workarounds in QGC.

The Solution:

On top of the existing Classic Bluetooth infrastructure, additions to the BluetoothLink class and BluetoothSettings.qml UI to support Bluetooth LE comm link. Allowing for users to switch between Classic or BLE mode when adding new bluetooth comm link.

Where to find it: Application Settings → Comm Links → Add

Implemented by Holden Ramsey

Disconnected - Click to manually connect

Takeoff

Return

0.0 ft 0.0 mph 00:00:00

0.0 ft 0.0 mph 0.0 ft

Why did we need this?

1. Newer autopilots support BLE
2. Good for cheaper, short-range communication on low-cost or development UAVs/UGVs.
3. Supports an architecture where QGC connects to a MAVLink router via Wi-Fi/BLE, which then uses a higher-powered radio to connect to the drone.

Added libarchive

Before: QGC only supported xz & zlib archive formats.

libarchive library adds support for more archive formats such as zip, tar, cpio and many more. The library's behavior is consistent across operating systems.

This enables QGC to handle a dozen of archive formats supported by libarchive inherently in the app without the need for platform specific archive tools.

Why in this matters?

UAVs use long range, lower bandwidth radios - compression matters

Firmware, mission, & parameter updates

For developers: Onboard software updates (OTA), Image recognition model updates, payload updates, granularized terrain data, natively viewing images/videos on QGC, etc…

Unit Testing & Code Coverage

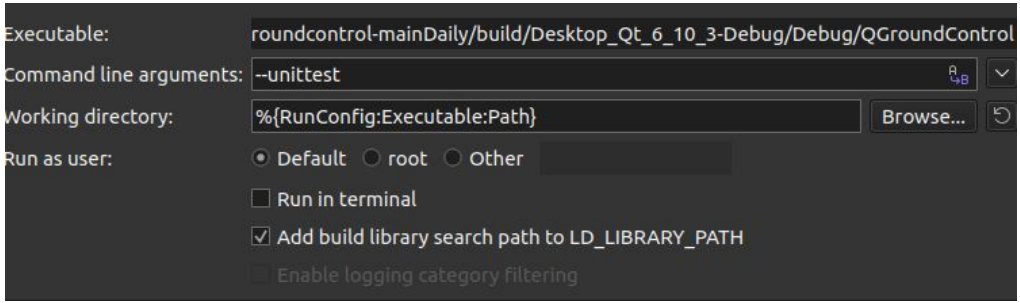
What is it?

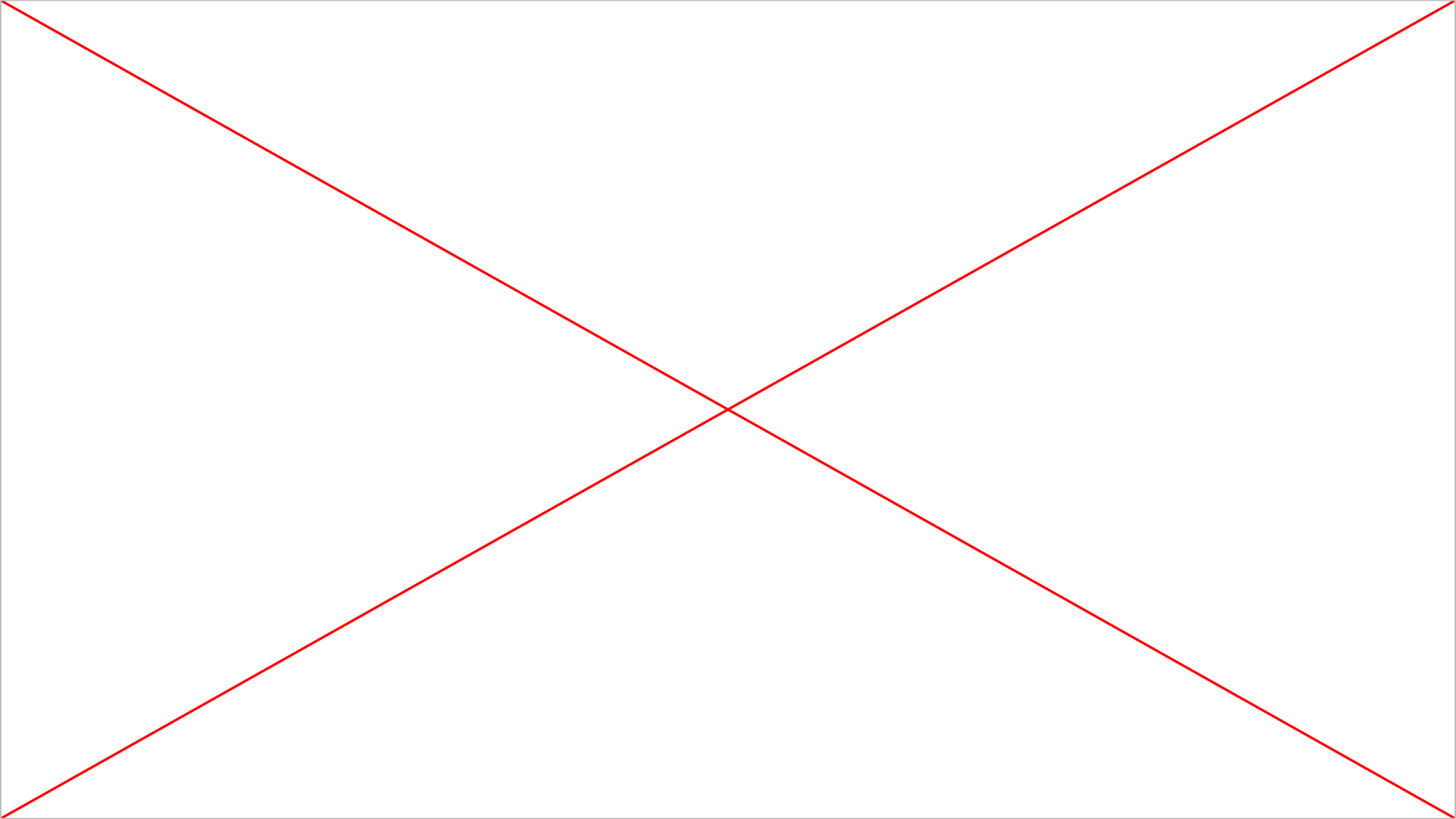
QGC is open source, so contributions are always welcome. To help facilitate sanity checks in development QGC has built in unit tests which act as a hard gate on contributions. Tests run inside the QGC process itself, not as a separate test binary i.e. the app launches under control of the test runner.

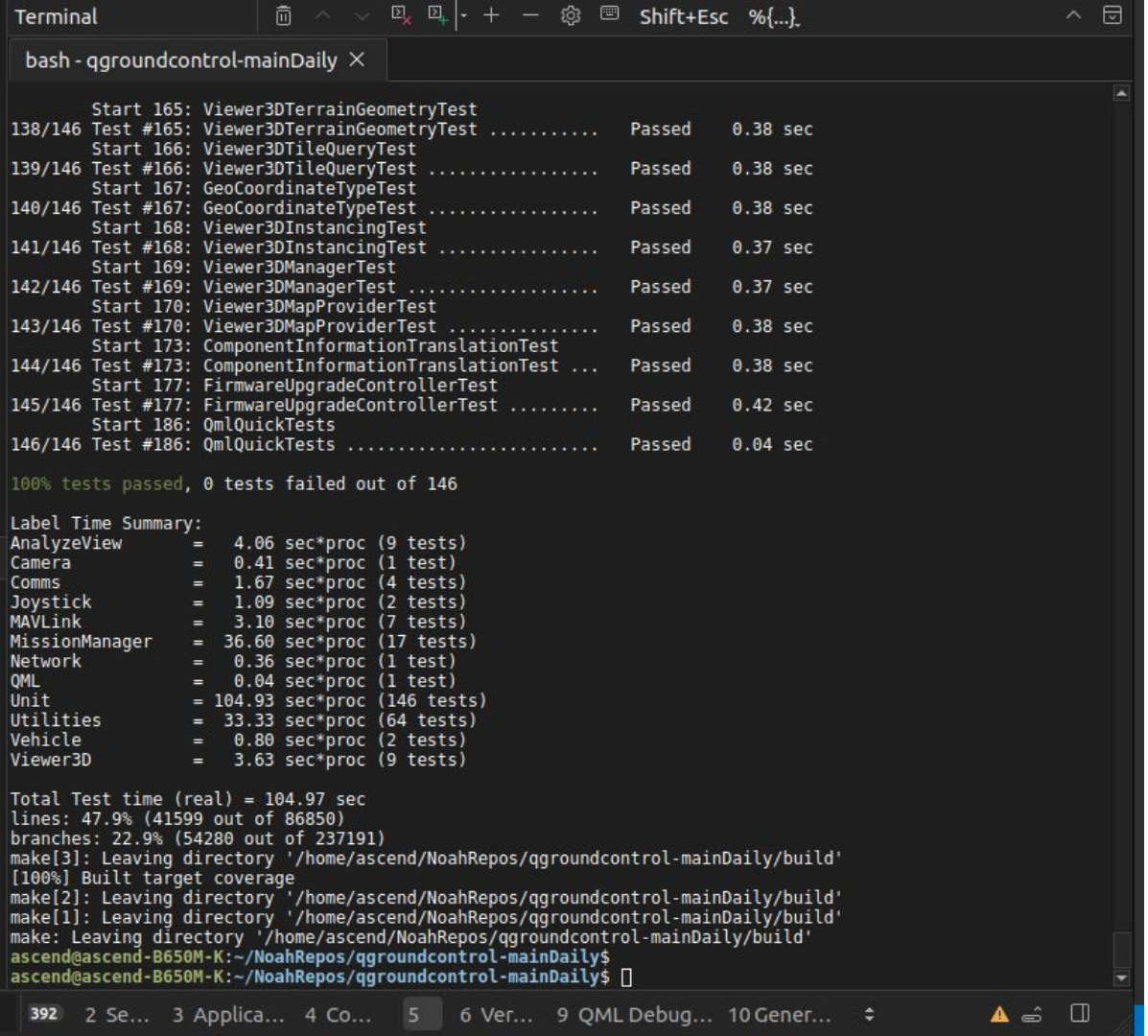
With 150+ test suites and code coverage tracking as well, contributions to QGC is more streamlined.

How to Build and Run

1. Build in debug mode with the QGC_BUILD_TESTING=ON in the CMake file
   a. -DQGC_BUILD_TESTING:BOOL=ON
   b. -DENABLE_COVERAGE:BOOL=ON
2. Run all tests: ./qgroundcontrol --unittest







Code coverage: from root directory

>cmake -B build
-DCMAKE_BUILD_TYPE=Debug
-DQGC_ENABLE_COVERAGE=ON

>cmake --build build --parallel

>cd build && ninja coverage

Coverage report in build directory:

/build/coverage.html

GCC Code Coverage Report

Directory: ./
Date: 2026-05-11 15:09:24
Legend: low: >= 0% | medium: >= 75.0% | high: >= 90.0%

	Exec	Total	Coverage
Lines:	41599	86850	47.9%
Branches:	54280	237191	22.9%

File	Lines		Branches	
src/ADSB/ADSB.h	100.0%	2 / 2	40.0%	4 / 10
src/ADSB/ADSBTCPLink.cc	64.0%	89 / 139	35.6%	83 / 233
src/ADSB/ADSBVehicle.cc	74.5%	35 / 47	40.2%	41 / 102
src/ADSB/ADSBVehicle.h	60.0%	6 / 10	-%	0 / 0
src/ADSB/ADSBVehicleManager.cc	55.4%	62 / 112	24.2%	46 / 190
src/ADSB/ADSBVehicleManager.h	100.0%	1 / 1	-%	0 / 0
src/API/QGCCorePlugin.cc	13.9%	28 / 201	7.4%	22 / 298
src/API/QGCCorePlugin.h	22.7%	5 / 22	0.0%	0 / 2
src/API/QGCOptions.cc	72.7%	8 / 11	10.0%	2 / 20
src/API/QGCOptions.h	8.3%	3 / 36	0.0%	0 / 2
src/API/QmlComponentInfo.cc	0.0%	0 / 7	0.0%	0 / 8
src/API/QmlComponentInfo.h	0.0%	0 / 4	-%	0 / 0
src/AnalyzeView/GeoTag/DataFlashParser.cc	0.0%	0 / 69	0.0%	0 / 144
src/AnalyzeView/GeoTag/DataFlashParser.h	0.0%	0 / 2	-%	0 / 0
src/AnalyzeView/GeoTag/ExifParser.cc	81.2%	39 / 48	34.3%	37 / 108
src/AnalyzeView/GeoTag/GeoTagController.cc	76.0%	456 / 600	35.7%	411 / 1152
src/AnalyzeView/GeoTag/GeoTagController.h	83.3%	20 / 24	-%	0 / 0
src/AnalyzeView/GeoTag/GeoTagData.h	100.0%	1 / 1	100.0%	4 / 4
src/AnalyzeView/GeoTag/GeoTagImageModel.cc	84.2%	80 / 95	62.9%	66 / 105
src/AnalyzeView/GeoTag/GeoTagImageModel.h	66.7%	2 / 3	50.0%	1 / 2
src/AnalyzeView/GeoTag/ULogParser.cc	86.5%	32 / 37	46.8%	58 / 124
src/AnalyzeView/GeoTag/ULogParser.h	0.0%	0 / 2	-%	0 / 0
src/AnalyzeView/MAVLinkConsole/MAVLinkConsoleController.cc	36.0%	64 / 178	18.4%	58 / 316
src/AnalyzeView/MAVLinkConsole/MAVLinkConsoleController.h	100.0%	2 / 2	-%	0 / 0
src/AnalyzeView/MAVLinkInspector/MAVLinkChartController.cc	37.9%	36 / 95	13.2%	15 / 114
src/AnalyzeView/MAVLinkInspector/MAVLinkChartController.h	77.8%	7 / 9	-%	0 / 0
src/AnalyzeView/MAVLinkInspector/MAVLinkInspectorController.cc	35.9%	52 / 145	32.3%	84 / 260
src/AnalyzeView/MAVLinkInspector/MAVLinkInspectorController.h	100.0%	6 / 6	-%	0 / 0
src/AnalyzeView/MAVLinkInspector/MAVLinkMessage.cc	27.8%	68 / 245	11.3%	43 / 380
src/AnalyzeView/MAVLinkInspector/MAVLinkMessage.h	90.0%	9 / 10	-%	0 / 0
src/AnalyzeView/MAVLinkInspector/MAVLinkMessageField.cc	25.6%	22 / 86	17.1%	13 / 76
src/AnalyzeView/MAVLinkInspector/MAVLinkMessageField.h	50.0%	4 / 8	-%	0 / 0
src/AnalyzeView/MAVLinkInspector/MAVLinkSystem.cc	97.4%	75 / 77	63.8%	51 / 80
src/AnalyzeView/MAVLinkInspector/MAVLinkSystem.h	83.3%	5 / 6	-%	0 / 0
src/AnalyzeView/OnboardLogs/OnboardLogController.cc	0.0%	0 / 372	0.0%	0 / 678
src/AnalyzeView/OnboardLogs/OnboardLogController.h	0.0%	0 / 7	-%	0 / 0
src/AnalyzeView/OnboardLogs/OnboardLogEntry.cc	0.0%	0 / 24	0.0%	0 / 10
src/AnalyzeView/OnboardLogs/OnboardLogEntry.h	0.0%	0 / 11	0.0%	0 / 10
src/AnalyzeView/OnboardLogsFtp/OnboardLogFtpController.cc	0.0%	0 / 257	0.0%	0 / 642
src/AnalyzeView/OnboardLogsFtp/OnboardLogFtpController.h	0.0%	0 / 4	-%	0 / 0

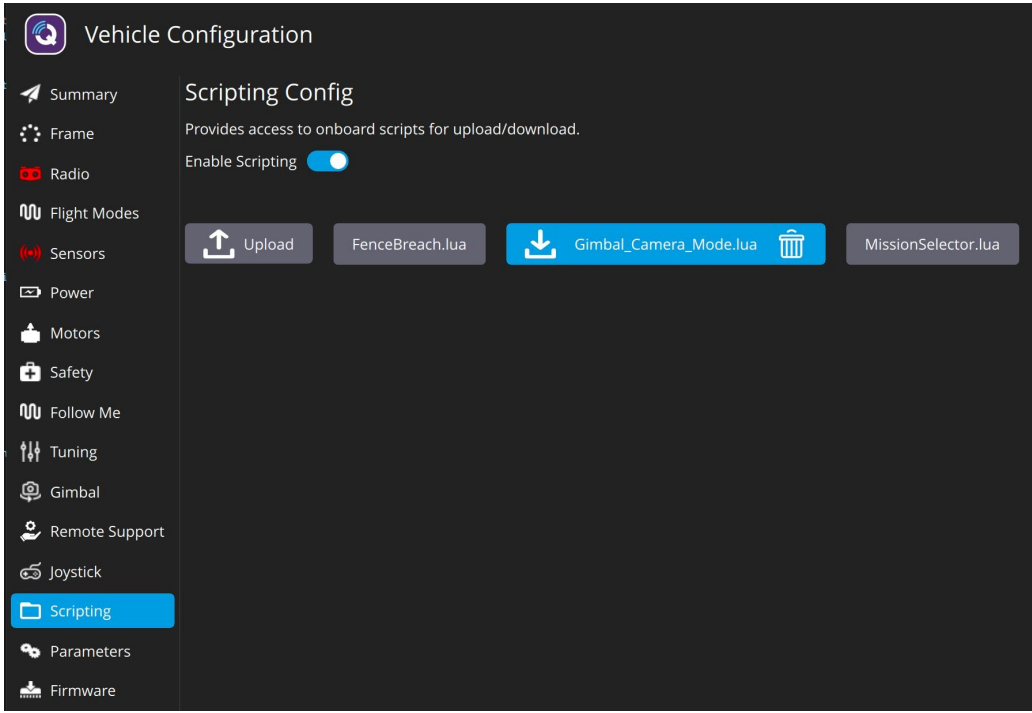
Lua Scripting Tab (ArduPilot Only)

What is it?

- Built in scripting interface inside QGC
- Run Lua scripts directly on a connected ArduPilot vehicle
- No external tools or custom builds needed

Why it Matters

- Useful for pre-flight checks
- Automated Workflows
- Customize MAVLink commands & prototyping new vehicle behaviors
- Streamline debugging



Where to find it: Vehicle Configuration → Scripting → Enable Scripting

Implemented by Donald Gagne

One App, Two Audiences

QGC Users: Pilots vs Developers - Advanced vs Simple Mode

What is it?

A global toggle in QGCCorePlugin that shows or hides advanced UI elements across QGroundControl.

When disabled, it strips away:

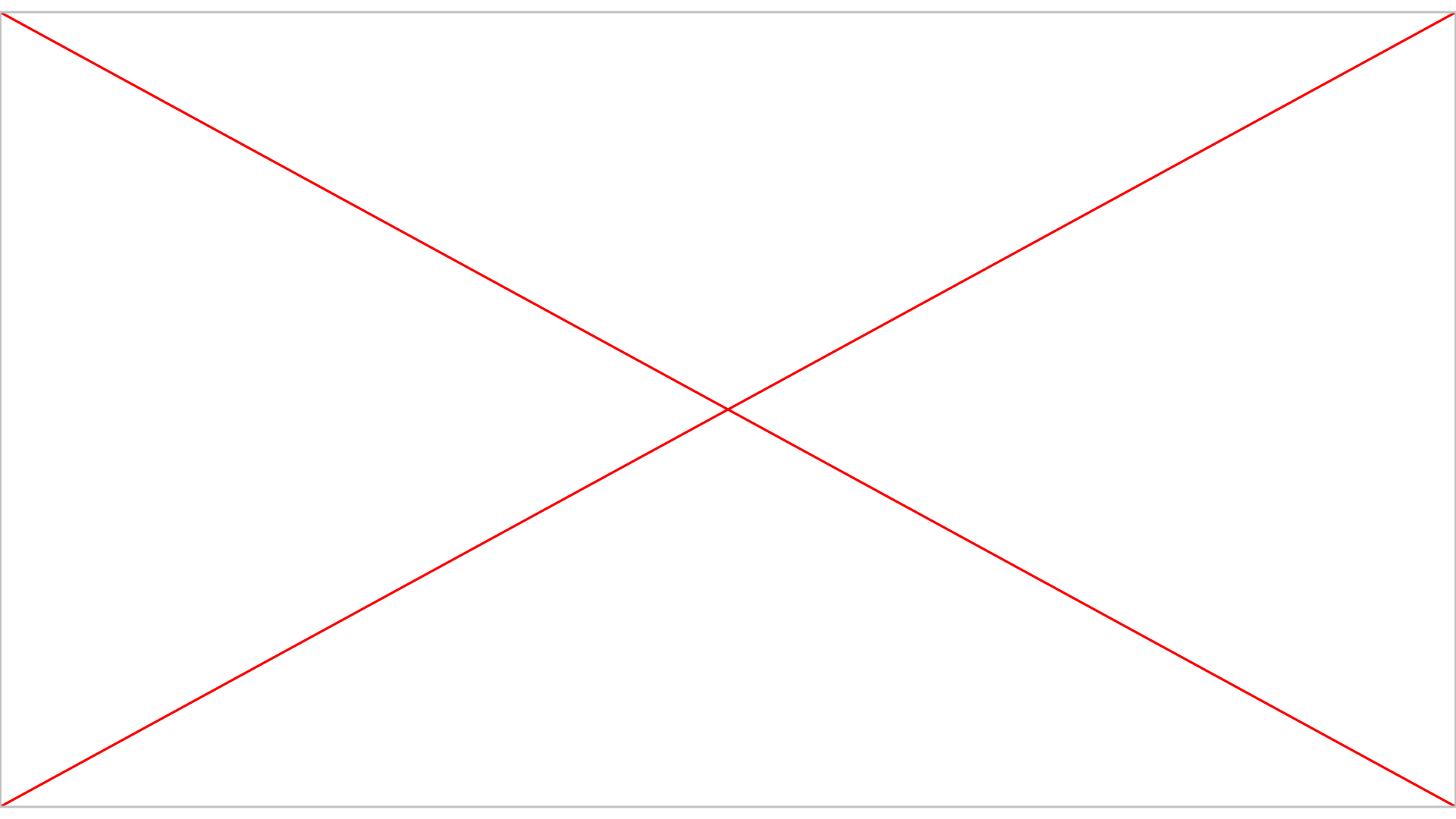
- Raw parameter browsing and force-save options
- Flight mode, power, and Remote ID configuration buttons
- The Analyze view (MAVLink Inspector, Log Download)
- Raw waypoint editing and mission camera settings

Why it matters

Custom builds can ship QGC to non-technical pilots without exposing vehicle internals that could cause misconfiguration or damage. The plugin architecture allows developers set the default state, customize the warning message, or gate access behind a password or other verification methods.

So the same codebase serves both a technical engineer doing deep vehicle setup and an end-user who just wants to start flying.

Togglable via Shift+click or clicking 5 times consecutively (click and hold on mobile) on the QGC verison in the main menu window



Soon: Rebuilding the video pipeline

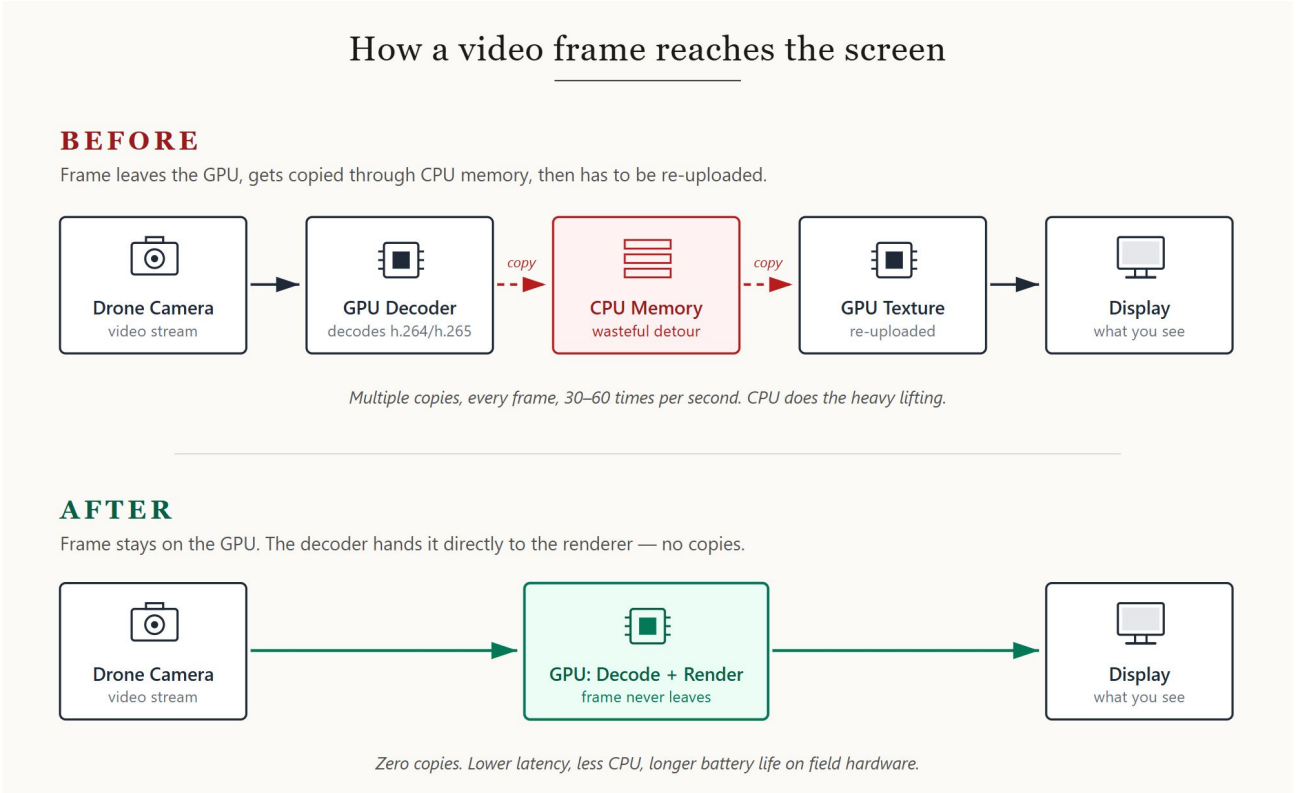
Two changes. Lower latency, less CPU, more headroom.

- Platform-native graphics backends: QGC used to force OpenGL on every platform. Now it uses what each OS actually wants — Metal on Apple, Direct3D on Windows, Vulkan on Linux and Android.
- Hardware buffers (zero-copy video)
    - Decoded frames stay on the GPU from decoder to display. No more round-trips through CPU memory.

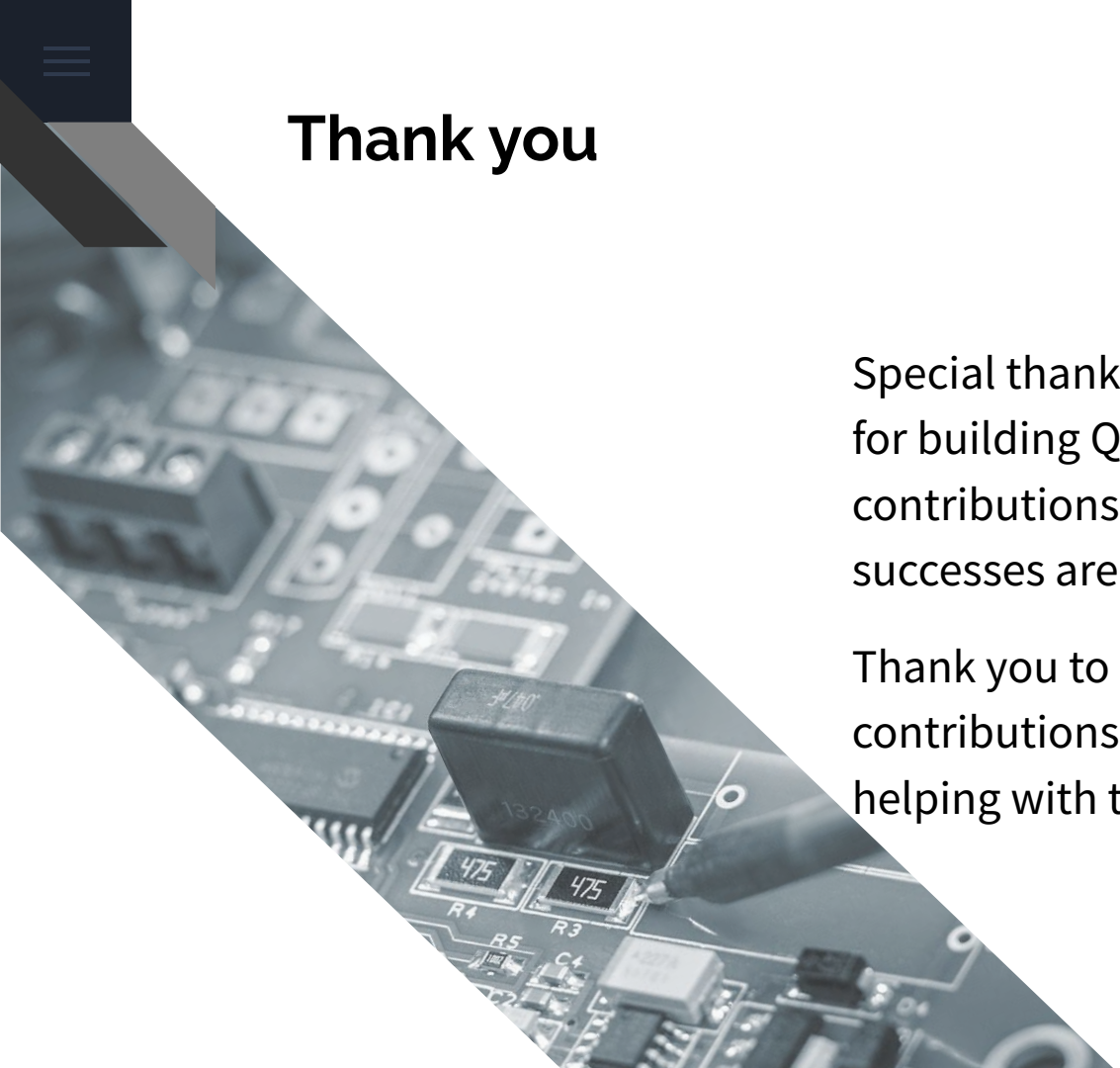
Why it matters

- Lower latency between drone and pilot
- Less CPU = less heat, longer battery on field tablets
- Headroom for higher resolution and multi-camera streams

Soon: Rebuilding the video pipeline



Thank you



Special thanks to Don Gagne (DonLakeFlyer) for building QGC and Holden Ramsey for their contributions to QGC. Any bugs are theirs. Any successes are mine.

Thank you to John Nomikos for his contributions as well, and Noah Sleeman for helping with this presentation.