# **QGroundControl Dev Guide**

This developer guide is the best source for information if you want to build, modify or extend [QGroundControl](http://qgroundcontrol.com/) (QGC). It shows how to obtain and build the source code, explains how QGC works, and provides guidelines for contributing code to the project.

This guide is for **developers**! To learn how to **use** *QGroundControl*, see the [User Guide](https://docs.qgroundcontrol.com/en/).

This guide is an active work in progress - information should be correct, but may not be complete! If you find that it is missing helpful information (or errors) please raise an [issue](https://github.com/mavlink/qgc-dev-guide/issues).

**Design Philosophy**

QGC is designed to provide a single codebase that can run across multiple OS platforms as well as multiple device sizes and styles.

The QGC user interface is implemented using [Qt QML](http://doc.qt.io/qt-5/qtqml-index.html). QML provides for hardware acceleration which is a key feature on lower powered devices such as tablets or phones. QML also provides features which allows us to more easily create a single user interface which can adapt itself to differing screen sizes and resolution.

The QGC UI targets itself more towards a tablet+touch style of UI than a desktop mouse-based UI. This make a single UI easier to create since tablet style UI also tends to work fine on desktop/laptops.

## **Support**

Development questions can be raised in the [QGroundControl Developer](http://discuss.px4.io/c/qgroundcontrol/qgroundcontrol-developers) discuss category or in the *QGroundControl* [Gitter](https://gitter.im/mavlink/qgroundcontrol) channel.

## **Contribution**

Information about contributing, including coding styles, testing and licenses can be found in [Code Submissions](https://dev.qgroundcontrol.com/en/contribute/).

We expect all contributors to adhere to the [QGroundControl code of conduct](https://github.com/mavlink/qgroundcontrol/blob/master/CODE_OF_CONDUCT.md). This code aims to foster an open and welcoming environment.

### **Translations**

We use [Crowdin](https://crowdin.com/) to make it easier to manage translation for both *QGroundControl* and the documentation.

The translation projects (and join links) are listed below:

* [QGroundControl](https://crowdin.com/project/qgroundcontrol) ([join](https://crwd.in/qgroundcontrol))
* [QGroundControl User Guide](https://crowdin.com/project/qgroundcontrol-user-guide) ([join](https://crwd.in/qgroundcontrol-user-guide))
* [QGroundControl Developer Guide](https://crowdin.com/project/qgroundcontrol-developer-guide) ([join](https://crwd.in/qgroundcontrol-developer-guide))

The PX4 Developer Guide contains additional information about the (common) docs/translation toolchain:

* [Documentation](https://dev.px4.io/en/contribute/docs.html)
* [Translation](https://dev.px4.io/en/contribute/docs.html)

## **License**

## *QGroundControl* source code is [dual-licensed under Apache 2.0 and GPLv3](https://github.com/mavlink/qgroundcontrol/blob/master/COPYING.md). For more information see: [Licenses](https://dev.qgroundcontrol.com/en/contribute/licences.html).

## **Governance**

The QGroundControl mission planner is hosted under the governance of the [Dronecode Project](https://www.dronecode.org/).



# Getting Started

This topic explains how to get the *QGroundControl* source code and build it either natively or within a *Vagrant* environment. It also provides information about optional or OS specific functionality.

## Daily Builds

If you just want to test (and not debug) a recent build of *QGroundControl* you can use the [Daily Build](https://docs.qgroundcontrol.com/en/releases/daily_builds.html). Versions are provided for all platforms.

## Source Code

Source code for *QGroundControl* is kept on GitHub here: <https://github.com/mavlink/qgroundcontrol>. It is [dual-licensed under Apache 2.0 and GPLv3](https://github.com/mavlink/qgroundcontrol/blob/master/COPYING.md).

To get the source files:

1. Clone the repo (or your fork) including submodules:

git clone <https://github.com/mavlink/qgroundcontrol.git> --recursive

1. Update submodules (required each time you pull new source code):

git submodule update

Github source-code zip files cannot be used because these do not contain the appropriate submodule source code. You must use git!

## Build QGroundControl

### Native Builds

*QGroundControl* builds are supported for macOS, Linux, Windows, iOS and Android. *QGroundControl* uses [Qt](http://www.qt.io/) as its cross-platform support library and uses [QtCreator](http://doc.qt.io/qtcreator/index.html) as its default build environment.

* **macOS:** v10.11 or higher
* **Ubuntu:** 64 bit, gcc compiler
* **Windows:** Vista or higher, [Visual Studio 2015 compiler](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#vs2015) (32 bit)
* **iOS:** 10.0 and higher
* **Android:** Jelly Bean (4.1) and higher. Standard QGC is built against ndk version 19.
* **Qt version:** 5.11.0 **(only)** (except for Ubuntu, which uses Qt 5.11.3)

For more information see: [Qt 5 supported platform list](http://doc.qt.io/qt-5/supported-platforms.html).

#### Install Visual Studio 2015 (Windows Only)

The Windows compiler can be found here: [Visual Studio 2015 compiler](https://visualstudio.microsoft.com/vs/older-downloads/) (32 bit)

When installing, you must minimally select all Visual C++ components as shown:

#### Visual Studio 2015 - Select all Visual C++ Components

#### Install Qt

You **need to install Qt as described below** instead of using pre-built packages from say, a Linux distribution, because *QGroundControl* needs access to private Qt headers.

To install Qt:

1. Download and run the [Qt Online Installer](http://www.qt.io/download-open-source)
   * **Ubuntu:**
     + Set the downloaded file to executable using: chmod +x.
     + Install to default location for use with **./qgroundcontrol-start.sh.** If you install Qt to a non-default location you will need to modify **qgroundcontrol-start.sh** in order to run downloaded builds.
2. In the installer *Select Components* dialog choose: 5.11.0 (on *Ubuntu* choose Qt 5.11.3).

Then install just the following components:

* + **Windows**: *MCVC 2015 32 bit*
  + **MacOS**: *macOS*
  + **Linux**: *Desktop gcc 64-bit*
  + All:
    - *Qt Charts* and *Qt Remote Objects (TP)*
    - *Android ARMv7* (to build Android)

1. Install Additional Packages (Platform Specific)
   * **Ubuntu:** sudo apt-get install speech-dispatcher libudev-dev libsdl2-dev
   * **Fedora:** sudo dnf install speech-dispatcher SDL2-devel SDL2 systemd-devel
   * **Arch Linux:** pacman -Sy speech-dispatcher
   * **Windows:** [USB Driver](http://www.pixhawk.org/firmware/downloads) to connect to Pixhawk/PX4Flow/3DR Radio
   * **Android:** [Qt Android Setup](http://doc.qt.io/qt-5/androidgs.html)

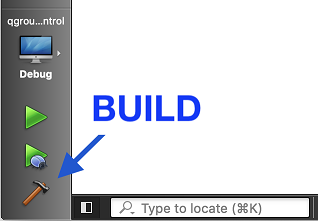
#### Building using Qt Creator

1. Launch *Qt Creator* and open the **qgroundcontrol.pro** project.
2. Select the appropriate kit for your needs:
   * **OSX:** Desktop Qt 5.11.0 clang 64 bit

iOS builds must be built using [XCode](http://doc.qt.io/qt-5/ios-support.html).

* + **Ubuntu:** Desktop Qt 5.11.3 GCC 64bit
  + **Windows:** Desktop Qt 5.11.0 MSVC2015 **32bit**
  + **Android:** Android for armeabi-v7a (GCC 4.9, Qt 5.11.0)

1. Build using the "hammer" (or "play") icons:



### Vagrant

[Vagrant](https://www.vagrantup.com/) can be used to build and run *QGroundControl* within a Linux virtual machine (the build can also be run on the host machine if it is compatible).

1. [Download](https://www.vagrantup.com/downloads.html) and [Install](https://www.vagrantup.com/docs/getting-started/) Vagrant
2. From the root directory of the *QGroundControl* repository run vagrant up
3. To use the graphical environment run vagrant reload

### Additional Build Notes for all Supported OS

* **Warnings as Errors:** Specifying CONFIG+=WarningsAsErrorsOn will turn all warnings into errors which breaks the build. If you are working on a pull request you plan to submit to github for consideration, you should always run with this setting turned on, since it is required for all pull requests.

Putting this line into a file called **user\_config.pri** in the top-level directory (same directory as **qgroundcontrol.pro**) will set this flag on all builds without interfering with the GIT history.

* **Parallel builds:** For non Windows builds, you can use the -j# option to run parellel builds.
* **Location of built files:** Individual build file results can be found in the build\_debug or build\_release directories. The built executable can be found in the debug or release directory.
* **If you get this error when running *QGroundControl***: /usr/lib/x86\_64-linux-gnu/libstdc++.so.6: version 'GLIBCXX\_3.4.20' not found., you need to either update to the latest *gcc*, or install the latest *libstdc++.6* using: sudo apt-get install libstdc++6.
* **Unit tests:** To run the [unit tests](https://word-edit.officeapps.live.com/contribute/unit_tests.html), build in debug mode with UNITTEST\_BUILD definition, and then copy deploy/qgroundcontrol-start.sh script into the debug directory before running the tests.

## Optional/OS-Specific Functionality

*QGroundControl* has functionality that is dependent on the operating system and libraries installed by the user. The following sections describe these features, their dependencies, and how to disable/alter them during the build process. These features can be forcibly enabled/disabled by specifying additional values to qmake.

+

### Opal-RT's RT-LAB Simulator

Integration with Opal-RT's RT-LAB simulator can be enabled on Windows by installing RT-LAB 7.2.4. This allows vehicles to be simulated in RT-LAB and communicate directly with QGC on the same computer as if the UAS was actually deployed. This support is enabled by default once the requisite RT-LAB software is installed. Disabling this can be done by adding DEFINES+=DISABLE\_RTLAB to qmake.

### XBee Support

*QGroundControl* can talk to XBee wireless devices using their proprietary protocol directly on Windows and Linux platforms. This support is not necessary if you're not using XBee devices or aren't using their proprietary protocol. On Windows, the necessary dependencies are included in this repository and no additional steps are required. For Linux, change to the libs/thirdParty/libxbee folder and run make;sudo make install to install libxbee on your system (uninstalling can be done with a sudo make uninstall). *qmake* will automatically detect the library on Linux, so no other work is necessary.

To disable XBee support you may add DEFINES+=DISABLE\_XBEE to *qmake*.

+

### Video Streaming

Check the [Video Streaming](https://github.com/mavlink/qgroundcontrol/tree/master/src/VideoStreaming) directory for further instructions.

## Building QGC Installation Files

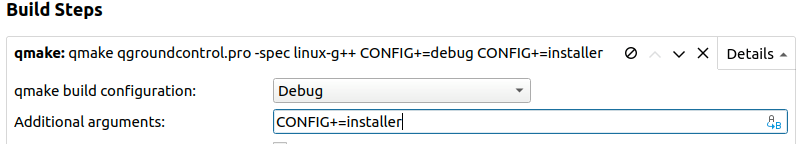
You can additionally create installation file(s) for *QGroundControl* as part of the normal build process.

On Windows you will need to first install [NSIS](https://sourceforge.net/projects/nsis/).

To add support for installation file creation you need to add CONFIG+=installer to your project file, or when you call *qmake*.

To do this in *Qt Creator*:

* Open **Projects > Build > Build Steps > qmake > Additional arguments**.
* Enter CONFIG+=installer as shown:



# **Communication Flow**

Description of the high level communication flow which takes place during a vehicle auto-connect.

* LinkManager always has a UDP link open waiting for a Vehicle heartbeat
* LinkManager detects a new known device (Pixhawk, SiK Radio, PX4 Flow) connected to computer
  + Creates a new SerialLink connected to the device
* Bytes comes through Link and are sent to MAVLinkProtocol
* MAVLinkProtocol converts the bytes into a MAVLink message
* If the message is a HEARTBEAT the MultiVehicleManager is notified
* MultiVehicleManager is notifed of the HEARTBEAT and creates a new Vehicle object based on the information in the HEARTBEAT message
* The Vehicle instantiates the plugins which match the vehicle type
* The ParameterLoader associated with the vehicle sends a PARAM\_REQUEST\_LIST to the vehicle to load params using the parameter protocol
* Once parameter load is complete, the MissionManager associated with the Vehicle requests the mission items from the Vehicle using the mission item protocol
* Once parameter load is complete, the VehicleComponents display their UI in the Setup view

# **Firmware Plugins**

Although the MAVLink spec defines a standard communication protocol to communicate with a vehicle. There are many aspects of that spec that at up for interpretation by the firmware developers. Because of this there are many cases where communication with a vehicle running one firmware must be slightly different to communication with a vehicle running a different firmware in order to accomplish the same task. Also each firmware may implement a subset of the MAVLink command set.

Another major issue is that the MAVLink spec does not cover vehicle configuration or a common parameter set. Due to this all code which relates to vehicle config ends up being firmware specific. Also any code which must refer to a specific parameter is also firmware specific.

Given all of these differences between firmware implementations it can be quite tricky to create a single ground station application that can support each without having the codebase degrade into a massive pile of if/then/else statements peppered everywhere based on the firmware the vehicle is using.

QGC uses a plugin architecture to isolate the firmware specific code from the code which is generic to all firmwares.

## **FirmwarePluginManager, FirmwarePlugin**

# **Class Hierarchy (high level)**

## **LinkManager, LinkInterface**

A "Link" in QGC is a specific type of communication pipe with the vehicle such as a serial port or UDP over WiFi. The base class for all links is LinkInterface. Each link runs on it's own thread and sends bytes to MAVLinkProtocol.

The LinkManager object keeps track of all open links in the system. LinkManager also manages automatic connections through serial and UDP links.

## **MAVLinkProtocol**

There is a single MAVLinkProtocol object in the system. It's job is to take incoming bytes from a link and translate them into MAVLink messages. MAVLink HEARTBEAT messages are routed to MultiVehicleManager. All MAVLink messages are routed to Vehicle's which are associated with the link.

## **MultiVehicleManager**

There is a single MultiVehicleManager object within the system. When it receives a HEARTBEAT on a link which has not been previously seen it creates a Vehicle object. `MultiVehicleManager also keeps tracks of all Vehicles in the system and handles switching from one active vehicle to another and correctly handling a vehicle being removed.

## **Vehicle**

The Vehicle object is the main interface through which the QGC code communicates with the physical vehicle.

Note: There is also a UAS object associated with each Vehicle which is a deprecated class and is slowly being phased out with all functionality moving to the Vehicle class. No new code should be added here.

## **FirmwarePlugin, FirmwarePluginManager**

The FirmwarePlugin class is used an the base class for firmware plugins. A firmware plugin contains the firmware specific code, such that the Vehicle object is clean with respect to it supporting a single standard interface to the UI.

FirmwarePluginManager is a factory class which creates a FirmwarePlugin instance based on the MAV\_AUTOPILOT/MAV\_TYPE combination for the Vehicle.

AutoPilotPlugin and AutoPilotPluginManager are deprecated class which also contains firmware specific code. All functionality in these are being moved to the newer FirmwarePlugin and FirmwarePluginManager implementations. No new code should be added here.

# **User Interface Design**

The main pattern for UI design in QGC is a UI page written in QML, many times communicated with a custom "Controller" written in C++. This follows a somewhat hacked variant of the MVC design pattern.

The QML code binds to information associated with the system through the following mechanisms:

* The custom controller
* The global QGroundControl object which provides access to things like the active Vehicle
* The FactSystem which provides access to parameters and in some cases custom Facts.

Note: Due to the complexity of the QML used in QGC as well as it's reliance on communication with C++ objects to drive the ui it is not possible to use the QML Designer provided by Qt to edit QML.

# **Multi-Device Design Pattern**

QGroundControl is designed to run on multiple device types from desktop to laptop to tablet to small phone sized screens using mouse and touch. Below is the description of how QGC does it and the reasoning behind it.

## **Efficient 1 person dev team**

The design pattern that QGC development uses to solve this problem is based around making new feature development quick and allowing the code base to be testable and maintained by a very small team (let's say 1 developer as the default dev team size). The pattern to achieve this is followed very strictly, because not following it will lead to slower dev times and lower quality.

Supporting this 1 person dev team concept leads to some tough decisions which not everyone may be happy about. But it does lead to QGC being released on many OS and form factors using a single codebase. This is something most other ground stations out their are not capable of achieving.

What about contributors you ask? QGC has a decent amount of contributors. Can't they help move things past this 1 person dev team concept. Yes QGC has quite a few contributors. But unfortunately they come and go over time. And when they go, the code they contributed still must be maintained. Hence you fall back to the 1 person dev team concept which is mostly what has been around as an average over the last three years of development.

## **Target Device**

The priority target for QGC UI design is a tablet, both from a touch standpoint and a screen size standpoint (think 10" Samsung Galaxy tab). Other device types and sizes may see some sacrifices of visuals and/or usability due to this decision. The current order when making priority based decisions is Tablet, Laptop, Desktop, Phone (any small screen).

### **Phone sized screen support**

At specified above, at this point smaller phone sized screens are the lowest level priority for QGC. More focus is put onto making active flight level displays such as the Fly view more usable. Less focus is placed on Setup related views such as Setup and Plan. Those specific view are tested to be functionally usable on small screens but they may be painful to use.

## **Development tools used**

### **Qt Layout controls**

QGC does not have differently coded UIs which are targeted to different screen sizes and/or form factors. In general it uses QML Layout capabilities to reflow a single set of QML UI code to fit different form factors. In some cases it provides less detail on small screen sizes to make things fit. But that is a simple visibility pattern.

### **FactSystem**

Internal to QGC is a system which is used to manage all of the individual pieces of data within the system. This data model is them connected to controls.

### **Heavy reliance on reusable controls**

QGC UI is developed from a base set of reusable controls and UI elements. This way any new feature added to a reusable control is now available throughout the UI. These reusable controls also connect to FactSystem Facts which then automatically provides appropriate UI.

## **Cons for this design pattern**

* The QGC user interface ends up being a hybrid style of desktop/laptop/tablet/phone. Hence not necessarily looking or feeling like it is optimized to any of these.
* Given the target device priority list and the fact that QGC tends to just re-layout the same UI elements to fit different form factors you will find this hybrid approach gets worse as you get farther away from the priority target. Hence small phone sized screens taking the worst hit on usability.
* The QGC reusable control set may not provide the absolute best UI in some cases. But it is still used to prevent the creation of additional maintenance surface area.
* Since the QGC UI uses the same UI code for all OSes, QGC does not follow the UI design guidelines specified by the OS itself. It has it's own visual style which is somewhat of a hybrid of things picked from each OS. Hence the UI looks and works mostly the same on all OS. Once again this means for example that QGC running on Android won't necessarily look like an android app. Or QGC running on an iPhone will not look or work like most other iPhone apps. That said the QGC visual/functional style should be understandable to these OS users.

## **Pros for this design pattern**

* It takes less time to design a new feature since the UI coding is done once using this hybrid model and control set. Layout reflow is quite capable in Qt QML and becomes second nature once you get used to it.
* A piece of UI can be functionally tested on only platform since the functional code is the same across all form factors. Only layout flow must be visually checked on multiple devices but this is fairly easily done using the mobile simulators. In most cases this is what is needed:
  + Use desktop build, resizing windows to test reflow. Just will generally cover a tablet sized screen as well.
  + Use a mobile simulator to visually verify a phone sized screen. On OSX XCode iPhone simulator works really well.
* All of the above are critical to keep our hypothetical 1 person dev team efficient and quality high.

## **Future directions**

* Raise phone (small screen) level prioritization to be more equal to Tablet. Current thinking is that this won't happen until a 3.3 release time frame (release after current one being worked on).

# **Font and Palette**

QGC has a standard set of fonts and color palette which should be used by all user interface.

import QGroundControl.Palette 1.0import QGroundControl.ScreenTools 1.0

## **QGCPalette**

This item exposes the QGC color palette. There are two variants of this palette: light and dark. The light palette is meant for outdoor use and the dark palette is for indoor. Normally you should never specify a color directly for UI, you should always use one from the palette. If you don't follow this rule, the user interface you create will not be capable of changing from a light/dark style.

## **QGCMapPalette**

The map palette is used for colors which are used to draw over a map. Due to the the different map styles, specifically satellite and street you need to have different colors to draw over them legibly. Satellite maps needs lighter colors to be seen whereas street maps need darker colors to be seen. The QGCMapPalette item provides a set of colors for this as well as the ability to switch between light and dark colors over maps.

## **ScreenTools**

The ScreenTools item provides values which you can use to specify font sizing. It also provides information on screen size and whether QGC is running on a mobile device.

# **User Interface Controls**

QGC provides a base set of controls for building user interface. In general they tend to be thin layers above the base QML Controls supported by Qt which respect the QGC color palette.

import QGroundControl.Controls 1.0

## **Qt Controls**

The following controls are QGC variants of standard Qt QML Controls. They provide the same functionality as the corresponding Qt controls except for the fact that they are drawn using the QGC palette.

* QGCButton
* QGCCheckBox
* QGCColoredImage
* QGCComboBox
* QGCFlickable
* QGCLabel
* QGCMovableItem
* QGCRadioButton
* QGCSlider
* QGCTextField

## **QGC Controls**

These custom controls are exclusive to QGC and are used to create standard UI elements.

* DropButton - RoundButton which drops out a panel of options when clicked. Example is Sync button in Plan view.
* ExclusiveGroupItem - Used a a base Item for custom controls which supports the QML ExclusiveGroup concept.
* QGCView - Base control for all top level views in the system. Provides support for FactPanels and displaying QGCViewDialogs and QGCViewMessages.
* QGCViewDialog - Dialog which pops out from the right side of a QGCView. You can specific the accept/reject buttons for the dialog as well as the dialog contents. Example usage is when you click on a parameter and it brings up the value editor dialog.
* QGCViewMessage - A simplified version of QGCViewDialog which allows you to specify buttons and a simple text message.
* QGCViewPanel - The main view contents inside of a QGCView.
* RoundButton - A round button control which uses an image as its inner contents.
* SetupPage - The base control for all Setup vehicle component pages. Provides a title, description and component page contents area
* **Fact System**

The Fact System provides a set of capabilities which standardizes and simplifies the creation of the QGC user interface.

## **Fact**

A Fact represents a single value within the system.

## **FactMetaData**

There is FactMetaData associated with each fact. It provides details on the Fact in order to drive automatic user interface generation and validation.

## **Fact Controls**

A Fact Control is a QML user interface control which connects to a Fact and it's FactMetaData to provide a control to the user to modify/display the value associated with the Fact.

## **FactGroup**

# **Top Level Views**

This section contains topics about the code for the top level views: settings, setup, plan and fly.

# **Settings View**

* Top level QML code is **AppSettings.qml**
* Each button loads a separate QML page

# **Setup View**

* Top level QML code implemented in **SetupView.qml**
* Fixed set of buttons/pages: Summary, Firmware
* Remainder of buttons/pages come from AutoPilotPlugin VehicleComponent list

# **Plan View**

* Top level QML code is in **MissionEditor.qml**
* Main visual UI is a FlightMap control
* QML communicates with MissionController (C++) which provides the view with the mission item data and methods

# **Fly View**

* Top level QML code is in **FlightDisplayView.qml**
* QML code communicates with MissionController (C++) for mission display
* Instrument widgets communicate with active Vehicle object
* Two main inner views are:
  + FlightDisplayViewMap
  + FlightDisplayViewVideo

# **File Formats**

This section contains topics about the file formats used/supported by *QGroundControl*

# **Parameters File Format**

# Onboard parameters for Vehicle 1

#

# # Vehicle-Id Component-Id Name Value Type

1 1 ACRO\_LOCKING 0 2

1 1 ACRO\_PITCH\_RATE 180 4

1 1 ACRO\_ROLL\_RATE 180 4

1 1 ADSB\_ENABLE 0 2

Above is an example of a parameter file with four parameters. The file can include as many parameters as needed.

Comments are preceded with a #.

This header: # MAV ID COMPONENT ID PARAM NAME VALUE describes the format for each row:

* Vehicle-Id Vehicle id
* Component-Id Component id for parameter
* Name Parameter name
* Value Parameter value
* Type Parameter type using MAVLink MAV\_PARAM\_TYPE\_\* enum values

A parameter file contains the parameters for a single Vehicle. It can contain parameters for multiple components within that Vehicle.

# Plan File Format

Plan files are stored in JSON file format and contain mission items and (optional) geo-fence and rally-points. Below you can see the top level format of a Plan file

This is "near-minimal" - a plan must contain at least one mission item. The plan fence and rally points are also used in modes when no mission is running.

{

"fileType": "Plan",

"geoFence": {

"circles": [

],

"polygons": [

],

"version": 2

},

"groundStation": "QGroundControl",

"mission": {

},

"rallyPoints": {

"points": [

],

"version": 2

},

"version": 1

}

The main fields are:

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version for this file. Current version is 1. |
| fileType | Must be "Plan". |
| groundStation | The name of the ground station which created this file (here *QGroundControl*) |
| [mission](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#mission) | The mission associated with this flight plan. |
| [geoFence](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#geofence) | (Optional) Geofence information for this plan. |
| [rallyPoints](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#rally_points) | (Optional) Rally/Safe point information for this plan |

## Mission Object

The structure of the mission object is shown below. The items field contains a comma-separated list of mission items (it must contain at least one mission item, as shown below). The list may be a mix of both [SimpleItem](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#mission_simple_item) and [ComplexItem](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#mission_complex_item) objects.

"mission": {

"cruiseSpeed": 15,

"firmwareType": 12,

"hoverSpeed": 5,

"items": [

{

"AMSLAltAboveTerrain": null,

"Altitude": 50,

"AltitudeMode": 0,

"autoContinue": true,

"command": 22,

"doJumpId": 1,

"frame": 3,

"params": [

15,

0,

0,

null,

47.3985099,

8.5451002,

50

],

"type": "SimpleItem"

}

],

"plannedHomePosition": [

47.3977419,

8.545594,

487.989

],

"vehicleType": 2,

"version": 2

},

The following values are required:

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version for the mission object. Current version is 2. |
| firmwareType | The firmware type for which this mission was created. This is one of the [MAV\_AUTOPILOT](https://mavlink.io/en/messages/common.html#MAV_AUTOPILOT) enum values. |
| vehicleType | The vehicle type for which this mission was created. This is one of the [MAV\_TYPE](https://mavlink.io/en/messages/common.html#MAV_TYPE) enum values. |
| cruiseSpeed | The default forward speed for Fixed wing or VTOL vehicles (i.e. when moving between waypoints). |
| hoverSpeed | The default forward speed for multi-rotor vehicles. |
| items | The list of mission item objects associated with the mission . The list may contain either/both [SimpleItem](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#mission_simple_item) and [ComplexItem](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#mission_complex_item) objects. |
| plannedHomePosition | The planned home position is shown on the map and used for mission planning when no vehicle is connected. The array values shown above are (from top): latitude, longitude and AMSL altitude. |

The format of the simple and complex items is given below.

### SimpleItem - Simple Mission Item

A simple item represents a single MAVLink [MISSION\_ITEM](https://mavlink.io/en/messages/common.html#MISSION_ITEM) command.

{

"AMSLAltAboveTerrain": null,

"Altitude": 50,

"AltitudeMode": 0,

"autoContinue": true,

"command": 22,

"doJumpId": 1,

"frame": 3,

"params": [

15,

0,

0,

null,

47.3985099,

8.5451002,

50

],

"type": "SimpleItem"

}

The field mapping is shown below.

|  |  |
| --- | --- |
| **Key** | **Description** |
| type | SimpleItem for a simple item |
| AMSLAltAboveTerrain | Altitude value shown to the user. |
| Altitude |  |
| AltitudeMode |  |
| autoContinue | [MISSION\_ITEM](https://mavlink.io/en/messages/common.html#MISSION_ITEM).autoContinue |
| command | The command ([MAV\_CMD](https://mavlink.io/en/messages/common.html#MAV_CMD)) for this mission item - see [MISSION\_ITEM](https://mavlink.io/en/messages/common.html#MISSION_ITEM).command. |
| doJumpId | The target id for the current mission item in DO\_JUMP commands. These are auto-numbered from 1. |
| frame | [MAV\_FRAME](https://mavlink.io/en/messages/common.html#MAV_FRAME) (see [MISSION\_ITEM](https://mavlink.io/en/messages/common.html#MISSION_ITEM).frame) |
| params | [MISSION\_ITEM](https://mavlink.io/en/messages/common.html#MISSION_ITEM).param1,2,3,4,x,y,z (values depends on the particular [MAV\_CMD](https://mavlink.io/en/messages/common.html#MAV_CMD)). |

### Complex Mission Item

A complex item is a higher level encapsulation of multiple MISSION\_ITEM objects treated as a single entity.

There are currently three types of complex mission items:

* [Survey](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#survey)
* [Corridor Scan](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#corridor_scan)
* [Structure Scan](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#structure_scan)

#### Survey - Complex Mission Item

The object definition for a Survey complex mission item is given below.

{

"TransectStyleComplexItem": {

...

},

"angle": 0,

"complexItemType": "survey",

"entryLocation": 0,

"flyAlternateTransects": false,

"polygon": [

[

-37.75170619863631,

144.98414811224316

],

...

[

-37.75170619863631,

144.99457681259048

]

],

"type": "ComplexItem",

"version": 4

},

Complex items have these values associated with them:

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version number for this survey definition. Current version is 3. |
| type | ComplexItem (this is a complex item). |
| complexItemType | survey |
| [TransectStyleComplexItem](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#TransectStyleComplexItem) | The common base definition for Survey and CorridorScan complex items. |
| angle | The angle for the transect paths (degrees). |
| entryLocation | ? |
| flyAlternateTransects | If true, the vehicle will skip every other transect and then come back at the end and fly these alternates. This can be used for fixed wing aircraft whn the turnaround would be too acute for the vehicle to make the turn. |
| polygon | The polygon array which represents the polygonal survey area. Each point is a latitude, longitude pair for a polygon vertex. |

#### Corridor Scan

The object definition for a CorridorScan complex mission item is given below.

{

"CorridorWidth": 50,

"EntryPoint": 0,

"TransectStyleComplexItem": {

...

},

},

"complexItemType": "CorridorScan",

"polyline": [

[

-37.75234887156983,

144.9893624624168

],

...

[

-37.75491914850321, 144.9893624624168

]

],

"type": "ComplexItem",

"version": 2

},

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version for this CorridorScan definition. Current version is 3. |
| type | ComplexItem (this is a complex item). |
| complexItemType | CorridorScan |
| CorridorWidth | ? |
| EntryPoint | ? |
| [TransectStyleComplexItem](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#TransectStyleComplexItem) | The common base definition for Survey and CorridorScan complex items. |
| polyline | ? |

#### Structure Scan

The object definition for a StructureScan complex mission item is given below.

{

"Altitude": 50,

"CameraCalc": {

"AdjustedFootprintFrontal":25, "AdjustedFootprintSide": 25,

"CameraName":"Manual (no camera specs)",

"DistanceToSurface": 10,

"DistanceToSurfaceRelative": true,

"version": 1

},

"Layers": 1,

"StructureHeight": 25,

"altitudeRelative": true,

"complexItemType": "StructureScan",

"polygon": [

[

-37.753184359536355, 144.98879374063998

],

...

[

-37.75408368012594, 144.98879374063998

]

],

"type": "ComplexItem",

"version": 2

}

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version for this StructureScan definition. Current version is 2. |
| type | ComplexItem (this is a complex item). |
| complexItemType | StructureScan |
| Altitude | ? |
| [CameraCalc](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#CameraCalc) | ? |
| Layers | ? |
| StructureHeight | ? |
| altitudeRelative | true: altitude is relative to home, false: altitude is AMSL. |
| polygon | ? |

#### TransectStyleComplexItem

TransectStyleComplexItem contains the common base definition for [survey](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#survey) and [CorridorScan](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#CorridorScan) complex items.

"TransectStyleComplexItem": {

"CameraCalc": {

...

},

"CameraTriggerInTurnAround": true,

"FollowTerrain": false,

"HoverAndCapture": false,

"Items": [

...

],

"Refly90Degrees": false,

"TurnAroundDistance": 10,

"VisualTransectPoints": [

[

-37.75161626657736,

144.98414811224316

],

...

[

-37.75565155437309,

144.99438539496475

]

],

"version": 1

},

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version for this TransectStyleComplexItem definition. Current version is 1. |
| [CameraCalc](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#CameraCalc) | ? |
| CameraTriggerInTurnAround | ? (boolean) |
| FollowTerrain | ? (boolean) |
| HoverAndCapture | ? (boolean) |
| Items | ? |
| Refly90Degrees | ? (boolean) |
| TurnAroundDistance | The distance to fly past the polygon edge prior to turning for the next transect. |
| VisualTransectPoints | ? |

##### CameraCalc

The CameraCalc contains camera information used for a survey, corridor or structure scan.

"CameraCalc": {

"AdjustedFootprintFrontal": 272.4,

"AdjustedFootprintSide": 409.2,

"CameraName": "Sony ILCE-QX1",

"DistanceToSurface": 940.6896551724138,

"DistanceToSurfaceRelative": true,

"FixedOrientation": false,

"FocalLength": 16,

"FrontalOverlap": 70,

"ImageDensity": 25,

"ImageHeight": 3632,

"ImageWidth": 5456,

"Landscape": true,

"MinTriggerInterval": 0,

"SensorHeight": 15.4,

"SensorWidth": 23.2,

"SideOverlap": 70,

"ValueSetIsDistance": false,

"version": 1

},

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version for this CameraCalc definition. Current version is 1. |
| AdjustedFootprintFrontal | ? |
| AdjustedFootprintSide | ? |
| DistanceToSurface | ? Units? |
| DistanceToSurfaceRelative | ? |
| CameraName | Name of camera being used (must correspond to one of the cameras known to *QGroundControl* or: Manual (no camera specs) for manual setup, Custom Camera for a custom setup. The keys listed after this point are not specified for a "Manual" camera definition. |
| FixedOrientation | ? (boolean) |
| FocalLength | Focal length of camera lens in millimeters. |
| FrontalOverlap | Percentage of frontal image overlap. |
| ImageDensity | ? |
| ImageHeight | Image height in px |
| ImageWidth | Image width in px |
| Landscape | true: Camera installed in landscape orientation on vehicle, false: Camera installed in portrait orientation on vehicle. |
| MinTriggerInterval | ? |
| SensorHeight | Sensor height in millimeters. |
| SensorWidth | Sensor width in millimeters. |
| SideOverlap | Percentage of side image overlap. |
| ValueSetIsDistance | ? (boolean) |

## GeoFence

Geofence information is optional. The plan can contain an arbitrary number of geofences defined in terms of polygons and circles.

The minimal definition is shown below.

+

"geoFence": {

"circles": [

],

"polygons": [

],

"version": 2

},

The fields are:

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version number for the geofence plan format. The documented version is 2. |
| [circles](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#circle_geofence) | List containing circle geofence definitions (comma separated). |
| [polygons](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=ko-KR&rs=ko-KR&hid=CwUhQuVD+EGG5oLt1QIefg.0&WOPISrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD12DB2555512DF7%213516&wdnd=1&wdPreviousSession=83714564-3889-410b-9627-32c3783b4bd4&wdNewAndOpenCt=1555560597880&wdo=4&wdOrigin=wacFileNew&wdPreviousCorrelation=872e9839-5b42-469a-9b10-dbb24bb70bf8&wde=docx&sc=host%3D%26qt%3DFolders&mscc=1&wdp=0#polygon_geofence) | List containing polygon geofence definitions (comma separated). |

### Circle Geofence

Each circular geofence is defined in a separate item, as shown below (multiple comma-separated items can be defined). The items define the centre and radius of the circle, and whether or not the specific geofence is activated.

+

{

"circle": {

"center": [

47.39756763610029,

8.544649762407738

],

"radius": 319.85

},

"inclusion": true,

"version": 1

}

The fields are:

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version number for the geofence "circle" plan format. The documented version is 1. |
| circle | The definition of the circle. Includes centre (latitude, longitude) and radisu as shown above. |
| inclusion | Whether or not the geofence is enabled (true) or disabled. |

### Polygon Geofence

Each polygon geofence is defined in a separate item, as shown below (multiple comma-separated items can be defined). The geofence includes a set of points defined with a clockwise winding (i.e. they must enclose an area).

+ {

"inclusion": true,

"polygon": [

[

47.39807773798406,

8.543834631785785

],

[

47.39983519888905,

8.550024648373267

],

[

47.39641100087146,

8.54499282423751

],

[

47.395590322265186,

8.539435808992085

]

],

"version": 1

}

],

"version": 2

}

The fields are:

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version number for the geofence "polygon" plan format. The documented version is 2. |
| polygon | A list of points for the polygon. Each point contains a latitude and longitude. The points are ordered in a clockwise winding. |
| inclusion | Whether or not the geofence is enabled (true) or disabled. |

## Rally Points

Rally point information is optional. The plan can contain an arbitrary number of rally points, each of which has a latitude, longitude, and altitude (above home position).

+

A definition with two points is shown below.

+

"rallyPoints": {

"points": [

[

47.39760401,

8.5509154,

50

],

[

47.39902017,

8.54263274,

50

]

],

"version": 2

}

The fields are:

|  |  |
| --- | --- |
| **Key** | **Description** |
| version | The version number for the rally point plan format. The documented version is 2. |
| points | A list of rally points. |

# **MAVLink Log Format**

*QGroundControl* allows you to generate plain MAVLink packet logs that can be replayed (with QGroundControl) to watch a mission again for analysis.

The format is binary:

* Byte 1-8: Timestamp in microseconds since Unix epoch as unsigned 64 bit integer
* Byte 9-271: MAVLink packet (263 bytes maximum packet length, not all bytes have to be actual data, the packet might be shorter. Includes packet start sign)

## **Debugging**

To check your data, open your written file in a hex editor. You should see after 8 bytes **0x55**. The first 8 bytes should also convert to a valid timestamp, so something either close to zero or around the number **1294571828792000** (which is the current Unix epoch timestamp in microseconds).

## **C++ Sketch for logging MAVLink**

The code fragment below shows how to implement logging using [C++ streams](http://www.cplusplus.com/reference/iostream/istream/) from the C++ standard library.

//write into mavlink logfil

ceconst int len = MAVLINK\_MAX\_PACKET\_LEN+sizeof(uint64\_t);

uint8\_t buf[len];

uint64\_t time = getSystemTimeUsecs();

memcpy(buf, (void\*)&time, sizeof(uint64\_t));

mavlink\_msg\_to\_send\_buffer(buf+sizeof(uint64\_t), msg);

mavlinkFile << buf << flush;

# **Developer Tools**

*QGroundControl* makes a number of tools available primarily for autopilot developers. These ease common developer tasks including setting up simulated connections for testing, and accessing the System Shell over MAVLink.

[Build the source in debug mode](https://github.com/mavlink/qgroundcontrol#supported-builds) to enable these tools.

Tools include:

* [**Mock Link**](https://dev.qgroundcontrol.com/en/tools/mock_link.html) (Daily Builds only) - Creates and stops multiple simulated vehicle links.
* [**Replay Flight Data**](https://docs.qgroundcontrol.com/en/app_menu/replay_flight_data.html) - Replay a telemetry log (User Guide).
* [**MAVLink Inspector**](https://docs.qgroundcontrol.com/en/app_menu/mavlink_inspector.html) - Display received MAVLink messages/values.
* [**MAVLink Analyzer**](https://docs.qgroundcontrol.com/en/app_menu/mavlink_analyzer.html) - Plot trends for MAVLink messages/values.
* [**Custom Command Widget**](https://docs.qgroundcontrol.com/en/app_menu/custom_command_widget.html) - Load custom/test QML UI at runtime.
* [**Onboard Files**](https://docs.qgroundcontrol.com/en/app_menu/onboard_files.html) - Navigate vehicle file system and upload/download files.
* [**HIL Config Widget**](https://docs.qgroundcontrol.com/en/app_menu/hil_config.html) - Settings for HIL simulators.
* [**MAVLink Console**](https://docs.qgroundcontrol.com/en/analyze_view/mavlink_console.html) (PX4 Only) - Connect to the PX4 nsh shell and send commands.

# **Mock Link**

*Mock Link* allows you to create and stop links to multiple simulated (mock) vehicles in *QGroundControl* debug builds.

The simulation does not support flight, but does allow easy testing of:

* Mission upload/download
* Viewing and changing parameters
* Testing most setup pages
* Multiple vehicle UIs

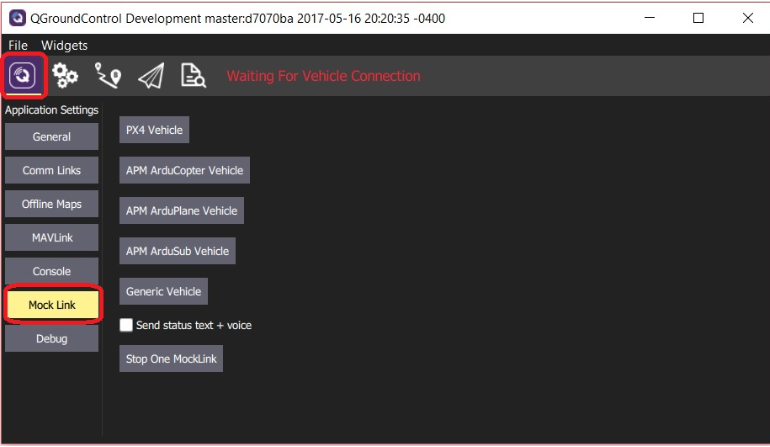
It is particularly useful for unit testing error cases for mission upload/download.

## **Using Mock Link**

To use *Mock Link*:

1. Create a debug build by [building the source](https://github.com/mavlink/qgroundcontrol#supported-builds).
2. Access *Mock Link* by selecting the *Application Settings* icon in the top toolbar and then **Mock Link** in the sidebar:

다음페이지1

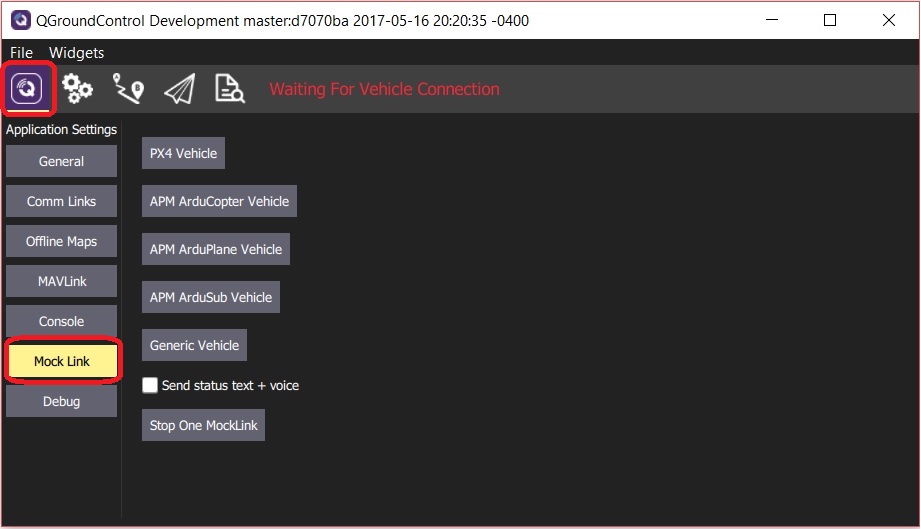


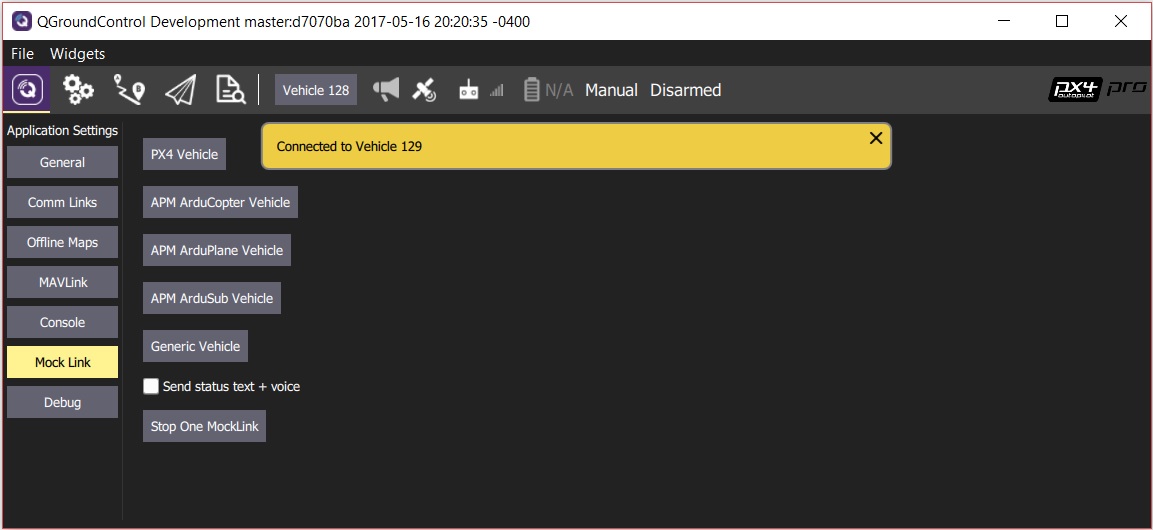
1. The buttons in the panel can be clicked to create a vehicle link of the associated type.
   * Each time you click a button a new connection will be created.
   * When there is more than one connection the multiple-vehicle UI will appear.

다음페이지에2

1. Click the **Stop one Mock Link** to stop the currently active vehicle.

Using *Mock Link* is then more or less the same as using any other vehicle, except that the simulation does not allow flight.

1

2

# **Command Line Options**

You can start *QGroundControl* with command line options. These are used to enable logging, run unit tests, and simulate different host environments for testing.

## **Starting QGroundControl with Options**

You will need to open a command prompt or terminal, change directory to where **qgroundcontrol.exe** is stored, and then run it. This is shown below for each platform (using the --logging:full option):

Windows Command Prompt:

cd "\Program Files (x86)\qgroundcontrol"

qgroundcontrol --logging:full

OSX Terminal app (**Applications/Utilities**):

cd /Applications/qgroundcontrol.app/Contents/MacOS/

./qgroundcontrol --logging:full

Linux Terminal:

./qgroundcontrol-start.sh --logging:full

## **Options**

The options/command line arguments are listed in the table below.

|  |  |
| --- | --- |
| **Option** | **Description** |
| --clear-settings | Clears the app settings (reverts *QGroundControl* back to default settings). |
| --logging:full | Turns on full logging. See [Console Logging](https://docs.qgroundcontrol.com/en/SettingsView/console_logging.html#logging-from-the-command-line). |
| --logging:full,LinkManagerVerboseLog,ParameterLoaderLog | Turns on full logging and turns off the following listed comma-separated logging options. |
| --logging:LinkManagerLog,ParameterLoaderLog | Turns on the specified comma separated logging options |
| --unittest:name | (Debug builds only) Runs the specified unit test. Leave off :name to run all tests. |
| --unittest-stress:name | (Debug builds only) Runs the specified unit test 20 times in a row. Leave off :name to run all tests. |
| --fake-mobile | Simulates running on a mobile device. |
| --test-high-dpi | Simulates running *QGroundControl* on a high DPI device. |

Notes:

* Unit tests are included in debug builds automatically (as part of *QGroundControl*). *QGroundControl* runs under the control of the unit test (it does not start normally).

# **Custom Builds**

A custom build allows third parties to create their own version of QGC in a way that allows them to easily keep up to date with changes which are being made in regular QGC. QGC has an archiecture built into it which allows customs builds to modofy and add to thje feature set of regular QGC.

**Keeping your custom build up to date**

## **Repository Setup**

* Create a new repository from main QGC repo. Do not clone, create a new repo, initializing it from the main QGC repo.
* You can now clone the above repo to do your work in and create pull requests from.
* In your clone create a remote called 'mavlink' which points back to the main QGC repo.
  + git remote add mavlink <https://github.com/mavlink/qgroundcontrol.git>

## **Upstream Merge**

We call the process of updating your custom build to the latest QGC bits and 'Upstream Merge'. Here is an example of how to do it:

* First make sure your local master is up to date with your own repos master.
* Create a branch to make all the changes to:
  + git checkout -b UpstreamMerge
* Pull in the latest bits from QGC:
  + git pull mavlink master
  + You'll get an editor to update merge comments. They are fine, just :q to exit.
* Now you need to update the resources in your custom build:
  + cd custom
  + python updateqrc.py
* Build it all to make sure there are no problems.
* You are now done. You can submit that as a Pull against your repo or however you want to get the changes into your main repo.

Note: This assume your custom build is based off of QGC master. If it is based off of a Stable branch just replace master with the stable branch name.

# **Code Submission**

This section contains topics about the contributing code, including coding style, testing and the format of pull requests.

QGroundControl (QGC) is [dual-licensed as Apache 2.0 and GPLv3](https://dev.qgroundcontrol.com/en/contribute/licences.html). All contributions have to be made under both licenses.

**Coding Style**

High level style information:

* Tabs expanded to 4 spaces
* Pascal/CamelCase naming conventions

The style itself is documents in the following example files:

* [CodingStyle.cc](https://github.com/mavlink/qgroundcontrol/blob/master/CodingStyle.cc)
* [CodingStyle.h](https://github.com/mavlink/qgroundcontrol/blob/master/CodingStyle.h)
* [CodingStyle.qml](https://github.com/mavlink/qgroundcontrol/blob/master/CodingStyle.qml)

# **Unit Tests**

*QGroundControl* (QGC) contains a set of unit tests that must pass before before a pull request will be accepted. The addition of new complex subsystems to QGC should include a corresponding new unit test to test it.

The full list of unit tests can be found in [UnitTestList.cc](https://github.com/mavlink/qgroundcontrol/blob/master/src/qgcunittest/UnitTestList.cc).

To run unit tests:

1. Build in debug mode with UNITTEST\_BUILD definition.
2. Copy the **deploy/qgroundcontrol-start.sh** script in the **debug** directory
3. Run *all* unit tests from the command line using the --unittest command line option. For Linux this is done as shown:

qgroundcontrol-start.sh --unittest

1. Run *individual* unit tests by specifying the test name as well: --unittest:RadioConfigTest. For Linux this is done as shown:

qgroundcontrol-start.sh --unittest:RadioConfigTest

**Pull Requests**

All pull requests go through the QGC CI build system which builds release and debug version. Builds will fail if there are compiler warnings. Also unit tests are run against supported OS debug builds

# **Licences**

## **QGroundControl License**

*QGroundControl* (QGC) is dual-licensed as Apache 2.0 and GPLv3. All contributions have to be made under both licenses. Users of the codebase are free to use it under either license.

*QGroundControl* licensing rules out the re-use of any copyleft (e.g. GPL) licensed code. All contributions must be original or from a compatible license (BSD 2/3 clause, MIT, Apache 2.0).

The dual approach is necessary to be able to offer *QGroundControl* through the iOS and Android app stores and offers the open source community choice.

### **Apache 2.0 License**

The [Apache 2.0](http://www.apache.org/licenses/LICENSE-2.0) License is a permissive license which allows QGC to be built and used in any environment, including proprietary applications. It allows QGC to be built for mobile app stores. When building with Apache 2.0 a commercial Qt license is required.

### **GPL v3 License**

The [GPL v3 License](http://www.gnu.org/licenses/gpl-3.0.en.html) is a strong copyleft license. When building QGC under this license the open source version of Qt can be used. Our licensing grants the permission to use a later version of the license, however, contributions have to be made under 3.0.

## **Documentation, Artwork, Images**

The QGroundControl documentation, artwork and images are licensed under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/).

## **See Also**

* [qgroundcontrol/COPYING.md](https://github.com/mavlink/qgroundcontrol/blob/master/COPYING.md)
* [qgroundcontrol/CONTRIBUTING.md](https://github.com/mavlink/qgroundcontrol/blob/master/CONTRIBUTING.md)
* [qgc-user-guide/LICENSE](https://github.com/mavlink/qgc-user-guide/blob/master/LICENSE)
* [qgc-dev-guide/LICENSE](https://github.com/mavlink/qgc-dev-guide/blob/master/LICENSE)