mavlink\_msg\_manual\_control\_pack

\_sendGCSHeartbeat

ParameterFactMetaData.xml存放了所有的参数变量

调试打印日志：

console.warn("Flight Modes are called 4")

// Link has been connected, but no Vehicle seen on link yet.

void linkConnected(LinkInterface\* link);

// Link disconnected, all vehicles on link should be gone as well.

void linkDisconnected(LinkInterface\* link);

// New vehicle has been seen on the link.

void linkActive(LinkInterface\* link, int vehicleId, int vehicleFirmwareType, int vehicleType);

Say: rc signal lost

I/QGC\_UsbDeviceJNI(27817): Say: rc signal regained

专业术语：

Horizontal situation indicator：水平位置指示器

疑问：

1-<parameter name="COM\_RC\_IN\_MODE" type="INT32" default="0">

<short\_desc>RC control input mode</short\_desc>

<long\_desc>The default value of 0 requires a valid RC transmitter setup. Setting this to 1 disables RC input handling and the associated checks. A value of 2 will generate RC control data from manual input received via MAVLink instead of directly forwarding the manual input data.</long\_desc>

<min>0</min>

<max>2</max>

-<values>

<value code="1">Disable RC Input Checks</value>

<value code="0">RC Transmitter</value>

<value code="2">Virtual RC by Joystick</value>

</values>

</parameter>

模式2是什么情况？

2为何在使用某个mavlink的link之前要link->\_setMavlinkChannel(i);

回答：当使用static inline void mavlink\_msg\_heartbeat\_send(mavlink\_channel\_t chan, uint8\_t type, uint8\_t autopilot, uint8\_t base\_mode, uint32\_t custom\_mode, uint8\_t system\_status)函数的时候，其中的chan是channel的缩写，用于选择发送的串口或者usb口。

下面这个函数就是检测哪个串口或者USB口可以使用的？

// Find a mavlink channel to use for this link

for (int i=0; i<32; i++) {

if (!(\_mavlinkChannelsUsedBitMask & 1 << i)) {

mavlink\_reset\_channel\_status(i);

link->\_setMavlinkChannel(i);

\_mavlinkChannelsUsedBitMask |= i << i;

channelSet = true;

break;

}

}

3 instrumentPanel.qgcView这个是如何显示roll pitch altitude等数据的？

// InstrumentSwipeView {

// id: \_valuesWidget

// width: parent.width

// qgcView: instrumentPanel.qgcView

// textColor: "white" //isSatellite ? "black" : "white"

// //backgroundColor: \_backgroundColor

// backgroundColor: "transparent"

// maxHeight: 600//\_availableValueHeight

# 业务逻辑相关

PX4的component有哪些？

# UI视图相关：

%%%%%%%%%%%%%%%%%%%%%根视图%%%%%%%%%%%%%%%%%%%%%%%%

MainWindowNative.qml地面站最高等级的window，可以认为是所有视图的基

{

Loader加载MainWindowInner.qml，id: *mainWindowInner*

}

MainWindowInner.qml在MainWindowNative.qml中加载的视图载体

{

MainToolBar.qml工具栏的加载id:*toolBar*

*flightViewr*为加载飞行视图FlightDisplayView.qml

*setupViewLoader*为加载设置视图SetupView.qml

*planViewLoader*为加载任务视图MissionEditor.qml

Loader MainWindowLeftPanel为主页面左边的选择面板MainWindowLeftPanel.qml id:*leftPanel*

Rectangle：System Message Area系统信息区域

Rectangle：Critical Message Area紧急信息区域

提供的函数：

*showFlyView*() {}

*showPlanView*() {}

*showSetupView*() {}

*showLeftMenu*() {}

*hideLeftMenu*() {}

*showMessageArea*() {}

*showPopUp*(dropItem, centerX) {}

}

%%%%%%%%%%%%%%%%%%%%%%根视图%%%%%%%%%%%%%%%%%%%%%%%

SetupView.qml设置界面，入口在工具栏中第二个Gears.svg图标

{

*showSummaryPanel(){加载*VehicleSummary.qml *}*

*showFirmwarePanel*(){}平台选择设置

*showJoystickPanel*(){}遥杆设置

*showParametersPanel*(){}参数设置

*showPX4FlowPanel*(){}PX4光流控制

*showVehicleComponentPanel*(vehicleComponent){}

}

MainToolBar.qml工具栏

{

Component：gpsInfo GPS信息

Component：*batteryInfo*电池信息

Component: *telemRSSIInfo*根据接收的信号强度指示距离

Row：Toolbar Row工具栏{ Hamburger、 Gears 、Plan 、PaperPlane }

Item{ id:*vehicleIndicators*飞机状态指示器

Loader:MainToolBarIndicators.qml

QGCLabel: COMMUNICATION LOST

QGCButtons: Disconnect

}

Rectangle进度条 id: *progressBar*

}

&&&&&&&&&&&&&&&&&&&& FlightDisplayView.qml &&&&&&&&&&&&&&&&&&&&&&&&&&&

FlightDisplayView.qml：飞行视图

{

Function:

setStates()

*setPipVisibility*(state)根据state设置视频的管道按钮是否可见

*px4JoystickCheck()*当设置了虚拟杆可用时需要检查\_activeVehicle等确保虚拟杆确实可用

Image id: *background*背景图VehicleNoseDownRotate.png

QGCViewPanel包括5样：

1 FlightDisplayViewMap地图 id: \_flightMap

2 左下角的像管道一样的视频进入按钮QGCPipable，点击1以后进入FlightDisplayViewVideo

3Loader id: *widgetsLoader*;此widgetsLoader.source = "FlightDisplayViewWidgets.qml"

4Loader id: *multiTouchItem；*虚拟杆Virtual Joystick；他会加载VirtualJoystick.qml

当组件加载完毕后，*widgetsLoader*开始加载FlightDisplayViewWidgets.qml

}

&&&&&&&&&&&&&&&&&&& FlightDisplayView.qml &&&&&&&&&&&&&&&&&&&&&&&&&&&&

FlightDisplayViewVideo.qml

{

}

VirtualJoystick.qml

{

Timer定时器，当有激活的飞机时候启动，开始发送2个虚拟遥杆的控制信号

Accelerometer加速度计读取元件反应了当前加速度计的最新读数 id: *accel1*，当onReadingChanged用来读取左右遥杆的控制量

MouseArea{ Image指纹图片id: *gyroImage*图片资源VehicleSummaryIcon.png }

QGCLabel：Flight Data，用Column类展示实时飞行数据

QGCLabel：Control Data，用Column类展示控制数据

MouseArea{ Image id:*joystickControl*背景VehicleLeftRotate.png，按下后激活左右2个虚拟杆}

MouseArea{ Image id:*gyroControl*背景为VehicleRightRotate.png，按下后激活虚拟指纹}

JoystickThumbPad共有3个，id: *leftStick、rightStick*（为*joystickControl所用*）*、*gyroStick（为*gyroControl所用*）

JoystickThumbPad 右边的虚拟遥杆id: *gyroStick*

}

JoystickThumbPad.qml虚拟遥杆控制控件，

{

背景图为JoystickBezelLight.png，

根据触点获得X、Y的具体坐标，

reCenter()重回定位原点，

根据*thumbDown*(touchPoints)获得按下的x、y值

连接信号与槽：

onXChanged:

onYChanged:

MultiPointTouchArea允许多点触碰并取得相关x、y信息

}

FlightDisplayViewWidgets.qml

{

Column id: gpsLockColumn ; 标签框No GPS Lock for Vehicle

QGCInstrumentWidgetAlternate：右边的中间可左右滑动选择的widget

包含如下选项

heading: \_heading

rollAngle: \_roll

pitchAngle: \_pitch

groundSpeedFact: \_groundSpeedFact

airSpeedFact: \_airSpeedFact

isSatellite: \_isSatellite

Vertical Tool Buttons：垂向的4个按钮

DropButton id: centerMapDropButton地图中心控制按钮（选择飞机是否在地图正中间等）

DropButton id: mapTypeButton地图图层选择按钮

RoundButton id: mapZoomPlus地图放大按钮

RoundButton id: mapZoomMinus地图缩小按钮

QGCInstrumentWidget：

}

QGCInstrumentWidget.qml

{

QGCCompassWidget罗盘计

QGCAttitudeWidget姿态计

}

MainWindowLeftPanel.qml

{ Item

{

onVisibleChanged: {根据可见性设置页面内容}

ParallelAnimation 动画秀出设置界面id: *\_\_animateShowDialog*

动画收起设置界面id: *\_\_animateHideDialog*

Rectangle半透明幕布，遮盖主页面id: *\_\_transparentSection*

Rectangle 顶部的分隔线 id: \_\_topSeparator

MouseArea点击后隐藏左边目录

Rectangle垂直方向上的分隔线id: *\_\_verticalSeparator*

Rectangle设置界面的左边选择区域id: *\_\_leftMenu*

{

QGCFlickable

{

QGCLabel： Preferences

QGCButton： General

Comm Links

Offline Maps

MavLink

Mock Link

Debug

}

}

Rectangle设置界面的右边主展示区域相关布局

{

Loader展示具体选项内容的容器，Loader的id: *\_\_rightPanel*

当选择左边的General/LinkSettings/OfflineMap/MavlinkSettings/MockLink/DebugWindow

任意一个按钮后加载相应的右半边的qml视图

Item取消按钮id: *closeButton*

}

}// Item

}

MissionEditor.qml

{

函数：

*updateMapToVehiclePosition*() {}重定位地图的中心与*Vehicle*重合

*loadFromVehicle*() {}从*Vehicle*加载任务项

*loadFromFile*() {}从文档中加载任务项目

*saveToFile*() {}保存任务到文档

*normalizeLat*(lat) {}规范经纬度的范围

*normalizeLon*(lon) {}规范经纬度的范围

*fitViewportToMissionItems*() {}从文档加载航点或者新添航点的时候能够在地图视图上面更新显示

*setCurrentItem*(sequenceNumber) {}设置当前的航点号Item

组件：

MissionController实现在MissionController.cc中，与MissionEditor.qml紧密相关，

有2个槽：onVisualItemsChanged、onNewItemsFromVehicle

ExclusiveGroup点击图层按钮后给出3个Map type选项，id: \_mapTypeButtonsExclusiveGroup

ExclusiveGroup左边四个圆形按钮，添加、加载、CenterMap、图层，id: \_dropButtonsExclusiveGroup

Component 在手机等移动客户端使用的，存储在文档中的任务的加载器 id: mobileFilePicker

Component 在手机等移动客户端使用的，保存任务到文档中 id: *mobileFileSaver*

Component 待考证id: *moveDialog*

QGCViewPanel

{

FlightMap

{

signal mapClicked(var coordinate)地图点击信号，携带位置信息

Component.onCompleted: *editorMap*.center = QGroundControl.flightMapPosition重合2个点

NumberAnimation添加航点号过程中的动画

MouseArea点击后根据像素点获取到经纬度信息

Rectangle 拖航点时候位置信息的重新获取id: *itemDragger*

{

onXChanged: liveDrag()

onYChanged: liveDrag()

function *liveDrag*() {}

function *clearItem*() {}

MouseArea{限定最大和最小拖动区域}

}

MapItemView添加绿色的航点在地图上

MapItemView添加白色的航点信息在右边的网格中

MapItemView添加退出航点的坐标信息项

Component航点退出组件id: *exitCoordinateComponent*

{

MissionItemIndicator

}

MapItemView添加简单的航点信息项

Component简单航点组件id: *missionItemComponent*

{

MissionItemIndicator

}

MissionLineView添加航点之间的连线

MapItemView添加飞机到地图中

Item进行选中的航点编辑id: *missionItemEditor*

{

onClicked点击后设置当前航点号

onRemove删除

onInsert插入

onMoveHomeToMapCenter移动Home点到地图中间

}

Column垂直工具按钮

{

添加航点按钮/更改形状按钮

任务同步按钮，与飞机同步、保存到文档、从文档加载，删除所有任务航点

中间图层按钮，选择地图的中间点是Home、Mission还是Vechicle

图层选择按钮，街景地图、卫星地图、复杂地图

}

RoundButton地图放大按钮

RoundButton地图缩小按钮

}

}

}

GeneralSettings.qml

# 整个代码框架细节解析：

QGCToolbox：This is used to manage all of our top level services/tools用来管理所有顶层的服务或者工具。

QGCQuickWidget：Subclass of QQuickWidget which injects Facts and the Pallete object into the QML context.是QQuickWidget的子类，用来插入Facts 和 Pallete对象到QML的上下文

QGCQmlWidgetHolder：This is used to create widgets which are implemented in QML.用来创建在QML中实现的widget

QGCPalette：QGCPalette is used by QML ui to bind colors to the QGC pallete. The implementation is similar to the QML SystemPalette and should be used in the same way. Refer to that documentation for details. The one difference is that QGCPalette also supports a light and dark theme which you can switch between. QGCPalette所使用的QML用户界面的颜色绑定到QGC调色板。该类的实现类似于QML SystemPalette并应以同样的方式使用。请参考该文档的详细信息。唯一的区别是，QGCPalette还支持亮丽与黑暗的主题，你可以在两者之间切换。

QGCMessageBox：Subclass of QMessageBox which re-implements the static public functions. There are two reasons for this: 1) The QMessageBox implementation on OSX does now show the title string. This leads to message boxes which don't make much sense. So on OSX we set title to text and text to informative text. 2) If parent is NULL, we set parent to MainWindow::instance该类是QMessageBox的子类，重新实现了其静态public函数。有两个原因：1）在OSX苹果操纵系统，QMessageBox提示实现并不显示标题字符串。这导致消息框没有太大的意义。因此，在OSX我们设置标题的文本和文本信息文本。2）如果父母为NULL，我们设置家长的MainWindow::实例

QGCMapPalette：QGCMapPalette is a variant of QGCPalette which is used to hold colors used for display over the map control. Since the coloring of a satellite map differs greatly from the coloring of a street map you need to be able to switch between sets of color based on map type. QGCMapPalette是QGCPalette的变体，它是用来存放用于显示在地图控件的颜色。由于一个卫星地图的着色从街道地图的着色有很大不同需要能够根据地图类型集的色彩之间切换。

QGCLoggingCategory：这是对于q日志记录类别一QGC特定的更换。这将注册类别名称到一个全局列表。它的用法是一样的\_LOGGING\_CATEOGRY。This is a QGC specific replacement for Q\_LOGGING\_CATEGORY. It will register the category name into a global list. It's usage is the same as \_LOGGING\_CATEOGRY.

QGCGeo：坐标转换的工具类，大地坐标系和北东地坐标系之间的转换

QGCFileDownload：文档下载类，可以通过http地址下载远程的文档或者本地的文档

QGCFileDialog：文档对话框，用于让用户选择文本（视频，或者加载waypoint文档）

QGCDockWidget：QGCQmlWidgetHolder的基类，用来创建在QML中实现的widget，传入title 和action

QGCConfig：配置文件，存放设置版本号以及应用名和组织名。

QGCApplication：This class is started by the main method and provides the central management unit of the groundstation application. 这个类由主函数调用，提供了地面站应用程序的中央管理单元。

{

公共函数：

标志位控制，

飞行数据保存，

进行单元测试，

风格选择（黑还是亮）

报告参数丢失，

显示消息，

显示手机图形界面

测试分辨率（DEGUG模式下）

工具箱

检测蓝牙是否可用

上次已知的Home Position

单元测试函数：

初始化通用设置

初始化正常App启动

初始化单元测试

显示设置Firmware，参数，概要，飞机组件

公共槽：

在主线程上显示消息框（提示消息，警告消息，致命消息）

显示飞行视图

显示计划视图

显示设置视图

Qml尝试关闭

在主线程上保存临时飞行数据（移动设备上）

私有槽：

参数丢失的槽

信号：

风格改变

checkForLostLogFiles()调用MAVLinkProtocol线程上的槽，检查丢失的日志文档

私有函数和变量：

加载当前风格

产生QML的根对象

Qml的App引擎（移动设备）

运行单元测试

}

QGC：全局类，颜色定义，地面站的系统时间

MG：定义了一些公共变量（版本号，系统信息，时间和距离的单位换算，显示信息）

LogCompressor：日志压缩机，继承QThread

JsonHelper：Json的帮助类，用来帮助解析Json文档

HomePositionManager：内部定义了HomePosition类，并管理HomePositio的添加和删除以及保存等工作。

GAudioOutput：音频输出类，文字转换成语音，消声等

VehicleSetup文件夹：机载设置（PX4 FMU主控芯片 PX4 Flow boards光流控芯片 3DR Radio串口通信）

Bootloader：引导程序加载工具

FirmwareImage：该类支持加载英特尔6固件文档，例如.px4, .bin, .ihx.

FirmwareUpgradeController：为了FirmwareUpgrade.qml而写的控制器

JoystickConfigController：标定遥杆时候用的控制器

PX4FirmwareUpgradeThread：内部由PX4FirmwareUpgradeThreadController提供方法与bootloader交互。它所发出的命令由PX4FirmwareUpgradeThreadWorker接管，使得它运行在一个异步线程上。

SetupViewTest：点击进入测试设置视图按钮

VehicleComponent：把飞机的部件抽象成为一组可配置值和用户图形界面

Vehicle文件夹：单个飞机、多架飞机管理类

Vehicle：单架飞机所有的属性和方法

Vehicle(LinkInterface\* link,

int vehicleId,

MAV\_AUTOPILOT firmwareType,

MAV\_TYPE vehicleType,

FirmwarePluginManager\* firmwarePluginManager,

AutoPilotPluginManager\* autopilotPluginManager,

JoystickManager\* joystickManager);

MultiVehicleManager：

发出signal：添加飞机，移除飞机，激活飞机，参数OK的飞机改变，激活的飞机改变了，地面站HeartBeat使能改变；

接收槽：删除飞机阶段1和2，激活飞机阶段2，自主导航参数已经改变，处理地面站发送的HeartBeat，飞机HeartBeat信息

Fact：A Fact is used to hold a single value within the system.仿佛是存放系统内具有（完成）某种特定功能的struct.

FactGroup：Used to group Facts together into an object hierarachy.用来把Fact组成一个具有层级结构的对象

FactMetaData：Holds the meta data associated with a Fact. This is kept in a seperate object from the Fact itself since you may have multiple instances of the same Fact. But there is only ever one FactMetaData instance or each Fact.用来存放唯一的元数据，不随实例化Fact的多少而变化。

FactSystem:The components of the FactSystem are a Fact which holds an individual value. FactMetaData holds additional meta data associated with a Fact such as description, min/max ranges and so forth. The FactValidator object is a QML validator which validates input according to the FactMetaData settings. Client code can then use this system to expose sets of Facts to QML code. An example of this is the PX4ParameterMetaData onbject which is part of the PX4 AutoPilot plugin. It exposes the firmware parameters to QML such that you can bind QML ui elements directly to parameters.系统用来存放一个个单独Fact的值。FactMetaData用来存放与Fact额外的元数据相关的值。

JoystickConfigController.cc：遥杆标定控制器，标定左右两个杆的最大值、最小值等

{

成员函数：start(void) 停止标定}

# C++常见疑问：

Q\_INVOKABLE宏：在定义一个类的成员函数前加上Q\_INVOKABLE宏来修饰，就可以让该方法被元对象系统调用。例如：Q\_INVOKABLE QString getButtonAction(int button);

以class RadioComponentController : public FactPanelController为例，其中的

Q\_ENUMS(BindModes);

Q\_PROPERTY(int minChannelCount MEMBER \_chanMinimum CONSTANT)；

Q\_INVOKABLE void spektrumBindMode(int mode);

都是为了其能够在\*.qml中使用

# Mavlink通讯相关：

void Vehicle::\_sendMessageOnLink(LinkInterface\* link, mavlink\_message\_t message)

{

// Make sure this is still a good link

if (!link || !\_links.contains(link) || !link->isConnected()) {

return;

}

// Give the plugin a chance to adjust

\_firmwarePlugin->adjustMavlinkMessage(this, &message);

static const uint8\_t messageKeys[256] = MAVLINK\_MESSAGE\_CRCS;

mavlink\_finalize\_message\_chan(&message,\_mavlink->getSystemId(),\_mavlink->getComponentId(),link->getMavlinkChannel(),message.len, messageKeys[message.msgid]);

// Write message into buffer, prepending start sign

uint8\_t buffer[MAVLINK\_MAX\_PACKET\_LEN];

int len = mavlink\_msg\_to\_send\_buffer(buffer, &message); //len代表不同命令的发送长度=MAVLINK\_NUM\_NON\_PAYLOAD\_BYTES + Length of payload

link->writeBytes((const char\*)buffer, len);

\_messagesSent++;

emit messagesSentChanged();

}

mavlink数据包的定义： <http://www.jianshu.com/p/23fddff60369>

按照通道号(channel)来解析一个uint8\_t 的byte字节，一般外层会用for循环一个字节一个字节的解析，根据return值MAVLINK\_HELPER判断此通道的状态，This function will return 0, 1 or

\* 2 (MAVLINK\_FRAMING\_INCOMPLETE, MAVLINK\_FRAMING\_OK or MAVLINK\_FRAMING\_BAD\_CRC)分别代表未解析完，解析完，解析错误。

@param uint8\_t chan表示

@return 0 if no message could be decoded, 1 on good message and CRC, 2 on bad CRC

MAVLINK\_HELPER uint8\_t mavlink\_parse\_char(uint8\_t chan, uint8\_t c, mavlink\_message\_t\* r\_message, mavlink\_status\_t\* r\_mavlink\_status)

此处就是开启定时器，50毫秒更新一次经纬度；定时发送的消息可以参考这个代码

// Refresh timer

connect(\_refreshTimer, &QTimer::timeout, this, &Vehicle::\_checkUpdate);

\_refreshTimer->setInterval(UPDATE\_TIMER);

\_refreshTimer->start(UPDATE\_TIMER);

例子：更改模式的流程：

1. 上层界面

Component {

id: *flightModeMenuItemComponent*

MenuItem {

onTriggered: {

if(activeVehicle) {

activeVehicle.flightMode = *text*

}

}

}

}

当目录被点击以后触发activeVehicle.flightMode = *text*触发了setFlightMode()

class Vehicle : public FactGroup

{

Q\_PROPERTY(QString flightMode READ flightMode WRITE setFlightMode NOTIFY flightModeChanged)

void setFlightMode(const QString& flightMode);

}

void Vehicle::setFlightMode(const QString& flightMode)

{

uint8\_t base\_mode;

uint32\_t custom\_mode;

if (\_firmwarePlugin->*setFlightMode*(flightMode, &base\_mode, &custom\_mode)) {

// setFlightMode will only set MAV\_MODE\_FLAG\_CUSTOM\_MODE\_ENABLED in base\_mode, we need to move back in the existing

// states.

uint8\_t newBaseMode = \_base\_mode & ~MAV\_MODE\_FLAG\_DECODE\_POSITION\_CUSTOM\_MODE;

newBaseMode |= base\_mode;

mavlink\_message\_t msg;

mavlink\_msg\_set\_mode\_pack(\_mavlink->getSystemId(), \_mavlink->getComponentId(), &msg, id(), newBaseMode, custom\_mode);

qWarning() << "FirmwarePlugin::setFlightMode success, flightMode:" << flightMode;

sendMessage(msg);

} else {

qWarning() << "FirmwarePlugin::setFlightMode failed, flightMode:" << flightMode;

}

}

bool PX4FirmwarePlugin::*setFlightMode*(const QString& flightMode, uint8\_t\* base\_mode, uint32\_t\* custom\_mode)

{

\*base\_mode = 0;

\*custom\_mode = 0;

bool found = false;

for (size\_t i=0; i<sizeof(rgModes2Name)/sizeof(rgModes2Name[0]); i++) {

const struct Modes2Name\* pModes2Name = &rgModes2Name[i];

if (flightMode.compare(pModes2Name->name, Qt::CaseInsensitive) == 0) {

union px4\_custom\_mode px4\_mode;

px4\_mode.data = 0;

px4\_mode.main\_mode = pModes2Name->main\_mode;

px4\_mode.sub\_mode = pModes2Name->sub\_mode;

qWarning() << "flight main\_mode: " << px4\_mode.main\_mode<< " setFlightMode success, flightMode: " << px4\_mode.sub\_mode;

\*base\_mode = MAV\_MODE\_FLAG\_CUSTOM\_MODE\_ENABLED;

\*custom\_mode = px4\_mode.data;

found = true;

break;

}

}

if (!found) {

qWarning() << "Unknown flight Mode" << flightMode;

}

return found;

}

connect(link, &LinkInterface::bytesReceived, \_mavlinkProtocol, &MAVLinkProtocol::receiveBytes);

connect(\_mavlink, &MAVLinkProtocol::messageReceived, this, &Vehicle::\_mavlinkMessageReceived);

PX4AutoPilotPlugin

# Don Gagne的GitBook

# 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

# 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 previsouly 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.

Note: 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.