# PX4说明文档

## PX4开发文档适用范围

* 获得[最低限度的开发人员设置](https://docs.px4.io/main/en/dev_setup/config_initial.html)，[从源代码构建 PX4](https://docs.px4.io/main/en/dev_setup/building_px4.html)并部署在[众多受支持的自动驾驶仪](https://docs.px4.io/main/en/flight_controller/)上。
* 了解[PX4 系统架构](https://docs.px4.io/main/en/concept/architecture.html)和其他核心概念。【重要】
* 了解如何修改飞行堆栈和中间件：【重要】
  + 修改飞行算法并添加新的[飞行模式](https://docs.px4.io/main/en/concept/flight_modes.html)。
  + 支持新[机身](https://docs.px4.io/main/en/dev_airframes/)。
* 了解如何将 PX4 与新硬件集成：【重要】
  + 支持新的传感器和执行器，包括摄像头、测距仪等。
  + 修改 PX4 以在新的自动驾驶仪硬件上运行。
* [模拟](https://docs.px4.io/main/en/simulation/)、[测试](https://docs.px4.io/main/en/test_and_ci/)和[调试/记录](https://docs.px4.io/main/en/debug/)PX4。【重要】
* 与外部机器人 API 进行通信/集成。

## 基本设备

* 安全飞行员的 **远程控制**
  + Taranis Plus 遥控器（或同等产品）
* **开发计算机**
  + 配备 OSX 10.15 或更高版本的 MacBook Pro（2015 年初及更新机型）
  + 配备 Ubuntu Linux 18.04 或更高版本的 Lenovo Thinkpad 450 (i5)
* **地面控制站**（电脑或平板电脑）：
  + iPad（需要 Wifi 遥测适配器）
  + 任何 MacBook 或 Ubuntu Linux 笔记本电脑（可以是开发计算机）
  + Samsung Note 4 或同等设备（任何最新的 Android 平板电脑或手机，具有足够大的屏幕以有效运行*QGroundControl*）。
* **能够运行 PX4 的车辆**：
  + [获得预制车辆](https://docs.px4.io/main/en/complete_vehicles/)
  + [建立你自己的](https://docs.px4.io/main/en/airframes/)
* **护目镜**
* **Tether**（仅限多旋翼飞行器 - 用于更具风险的测试）

## 车辆配置

安装[QGroundControl 每日构建](https://docs.qgroundcontrol.com/master/en/qgc-user-guide/releases/daily_builds.html)对于**桌面操作系统**。

配置车辆：

1. [安装 PX4 固件](https://docs.px4.io/main/en/config/firmware.html#installing-px4-main-beta-or-custom-firmware)（包括您自己更改的“自定义”固件）。
2. [从机身参考](https://docs.px4.io/main/en/airframes/airframe_reference.html)中选择与您的车辆最匹配的[机身开始](https://docs.px4.io/main/en/config/airframe.html)。
3. [基本配置](https://docs.px4.io/main/en/config/)解释了如何执行基本配置。【重要】
4. [参数配置](https://docs.px4.io/main/en/advanced_config/parameters.html)解释了如何查找和修改各个参数。【重要】

## 构建PX4软件

1. 下载源代码

PX4 源代码存储在[PX4/PX4-Autopilot 的Github 上](https://github.com/PX4/PX4-Autopilot)存储库。

要将*最新*（“主”）版本安装到您​​的计算机上，请在终端中输入以下命令：

|  |
| --- |
| git clone https://github.com/PX4/PX4-Autopilot.git --recursive |

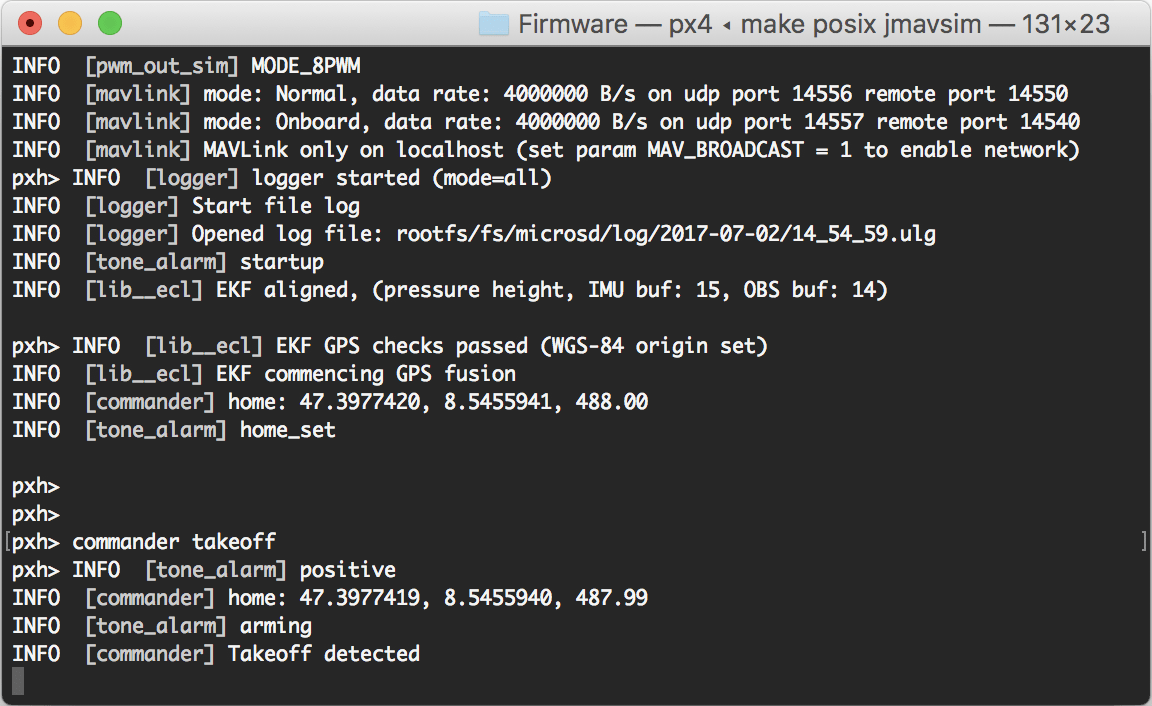
1. 第一次构建（使用jMAVSim模拟器）

首先，我们将使用控制台环境构建一个模拟目标。这使我们能够在转向实际硬件和 IDE 之前验证系统设置。

导航到**PX4-Autopilot**目录并使用以下命令启动[jMAVSim ：](https://docs.px4.io/main/en/sim_jmavsim/)

|  |
| --- |
| make px4\_sitl jmavsim |

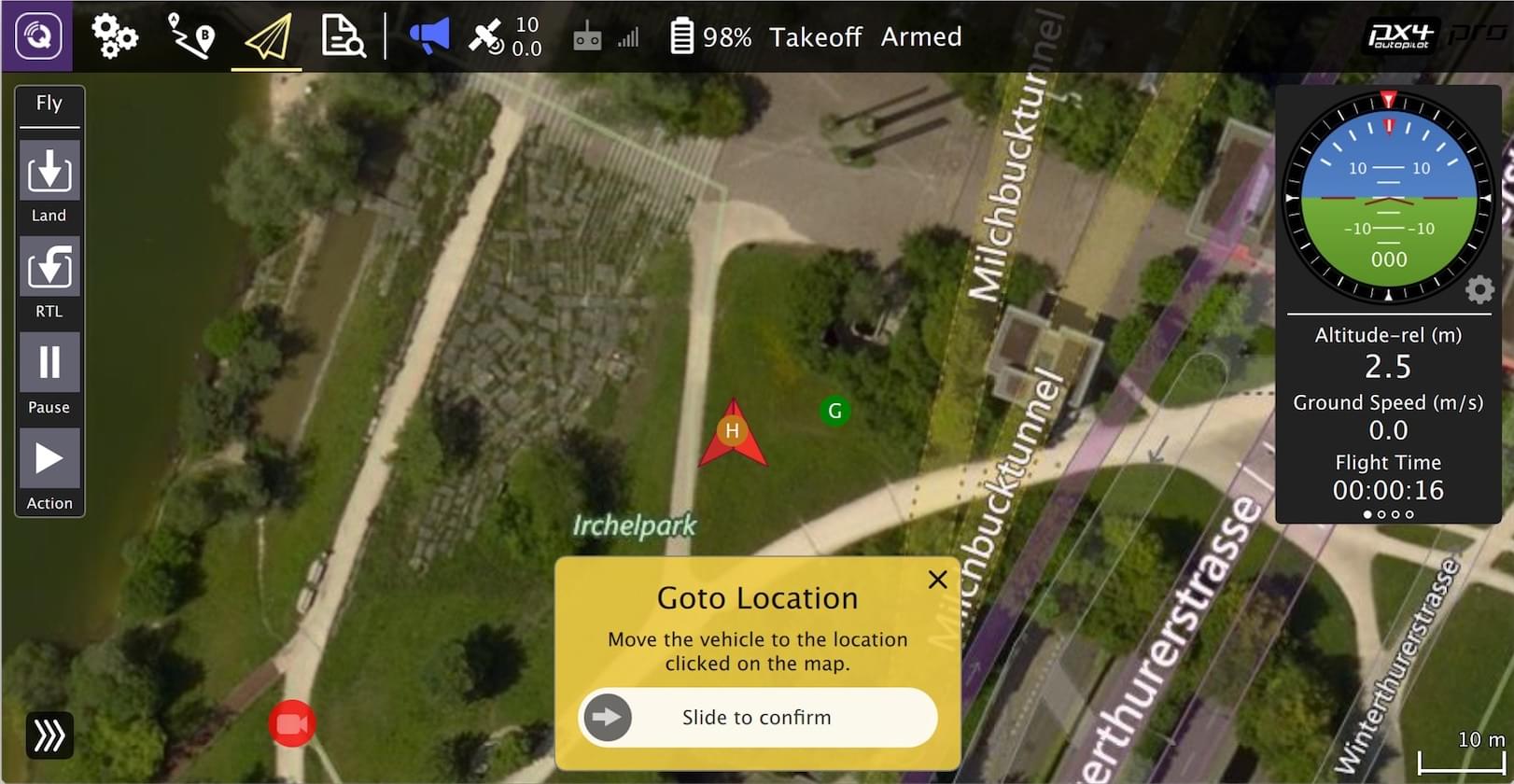
这将打开下面的 PX4 控制台：



【提示：您可能需要在继续之前启动*QGroundControl*，因为默认 PX4 配置需要在起飞前建立地面控制连接。可以[从这里下载](https://docs.qgroundcontrol.com/master/en/getting_started/download_and_install.html)。】

无人机可以通过打字着陆，并且可以通过执行**CTRL+C**（或输入）commander land来停止整个模拟，输入shutdown命令即可。

通过地面控制站进行模拟飞行更接近车辆的真实操作。在车辆飞行时（起飞飞行模式）单击地图中的某个位置并启用滑块。这将重新定位车辆。



【提示：PX4 可与许多其他[模拟器](https://docs.px4.io/main/en/simulation/)一起使用，包括[Gazebo](https://docs.px4.io/main/en/sim_gazebo_gz/)、[Gazebo Classic](https://docs.px4.io/main/en/sim_gazebo_classic/)和[AirSim](https://docs.px4.io/main/en/sim_airsim/)。这些也是从*make*开始的，例如构建时使用如下代码：

|  |
| --- |
| make px4\_sitl gazebo-classic |

】

1. 基于NuttX/Pixhawk的主板

3,1为NuttX构建

要构建基于 NuttX 或 Pixhawk 的板，请导航到**PX4-Autopilot**目录，然后make使用板的构建目标进行调用。例如，要构建[Pixhawk 4](https://docs.px4.io/main/en/flight_controller/pixhawk4.html)硬件，您可以使用以下命令：

|  |
| --- |
| cd PX4-Autopilot  make px4\_fmu-v5\_default |

成功的运行将以类似的输出结束：

|  |
| --- |
| -- Build files have been written to: /home/youruser/src/PX4-Autopilot/build/px4\_fmu-v4\_default  [954/954] Creating /home/youruser/src/PX4-Autopilot/build/px4\_fmu-v4\_default/px4\_fmu-v4\_default.px4 |

【笔记：构建得到的固件将存储在PX4-Autopilot/build/文件夹中】

构建目标的第一部分px4\_fmu-v4指示固件的目标飞行控制器硬件。在这种情况下，后缀\_default表示固件*配置*，例如支持或省略特定功能。

【笔记：后缀\_default是可选的。例如，make px4\_fmu-v5并px4\_fmu-v5\_default产生相同的固件。】

以下列表显示了[Pixhawk 标准](https://docs.px4.io/main/en/flight_controller/autopilot_pixhawk_standard.html)板的构建命令：

* [Holybro Pixhawk 6X (FMUv6X)](https://docs.px4.io/main/en/flight_controller/pixhawk6x.html) :make px4\_fmu-v6x\_default
* [Holybro Pixhawk 6C (FMUv6C)](https://docs.px4.io/main/en/flight_controller/pixhawk6c.html) :make px4\_fmu-v6c\_default
* [Holybro Pixhawk 6C 迷你 (FMUv6C)](https://docs.px4.io/main/en/flight_controller/pixhawk6c_mini.html)：make px4\_fmu-v6c\_default
* [Holybro Pix32 v6 (FMUv6C)](https://docs.px4.io/main/en/flight_controller/holybro_pix32_v6.html) :make px4\_fmu-v6c\_default
* [Holybro Pixhawk 5X (FMUv5X)](https://docs.px4.io/main/en/flight_controller/pixhawk5x.html) :make px4\_fmu-v5x\_default
* [Pixhawk 4 (FMUv5)](https://docs.px4.io/main/en/flight_controller/pixhawk4.html)：make px4\_fmu-v5\_default
* [Pixhawk 4 迷你 (FMUv5)](https://docs.px4.io/main/en/flight_controller/pixhawk4_mini.html)：make px4\_fmu-v5\_default
* [CUAV V5+ (FMUv5)](https://docs.px4.io/main/en/flight_controller/cuav_v5_plus.html) :make px4\_fmu-v5\_default
* [CUAV V5 纳米（FMUv5）](https://docs.px4.io/main/en/flight_controller/cuav_v5_nano.html)：make px4\_fmu-v5\_default
* [Pixracer（FMUv4）](https://docs.px4.io/main/en/flight_controller/pixracer.html)：make px4\_fmu-v4\_default
* [Pixhawk 3 Pro](https://docs.px4.io/main/en/flight_controller/pixhawk3_pro.html)：make px4\_fmu-v4pro\_default
* [迷你皮克斯霍克](https://docs.px4.io/main/en/flight_controller/pixhawk_mini.html)：make px4\_fmu-v3\_default
* [Pixhawk 2（立方体黑色）（FMUv3）](https://docs.px4.io/main/en/flight_controller/pixhawk-2.html)：make px4\_fmu-v3\_default
* [mRo Pixhawk (FMUv3)](https://docs.px4.io/main/en/flight_controller/mro_pixhawk.html) : make px4\_fmu-v3\_default(支持 2MB 闪存)
* [Holybro pix32 (FMUv2)](https://docs.px4.io/main/en/flight_controller/holybro_pix32.html) :make px4\_fmu-v2\_default
* [Pixfalcon（FMUv2）](https://docs.px4.io/main/en/flight_controller/pixfalcon.html)：make px4\_fmu-v2\_default
* [Dropix（FMUv2）](https://docs.px4.io/main/en/flight_controller/dropix.html)：make px4\_fmu-v2\_default
* [Pixhawk 1 (FMUv2)](https://docs.px4.io/main/en/flight_controller/pixhawk.html)：make px4\_fmu-v2\_default
* 具有 2 MB 闪存的 Pixhawk 1：make px4\_fmu-v3\_default

[各个飞行控制器板](https://docs.px4.io/main/en/flight_controller/)的文档中提供了非 Pixhawk NuttX 战斗控制器（以及所有其他板）的构建命令。

3.2上传固件

附加upload到 make 命令以通过 USB 将编译的二进制文件上传到自动驾驶仪硬件。例如：

|  |
| --- |
| make px4\_fmu-v4\_default upload |

成功运行将以以下输出结束：

|  |
| --- |
| Erase : [====================] 100.0%  Program: [====================] 100.0%  Verify : [====================] 100.0%  Rebooting.  [100%] Built target upload |

1. 故障排除

4.1一般构建错误

许多构建问题是由不匹配的子模块或不完全清理的构建环境引起的。更新子模块并执行以下操作distclean可以修复这些类型的错误：

|  |
| --- |
| git submodule update --recursive  make distclean |

4.2其他错误处理方式

见如下网址罗列项目：

|  |
| --- |
| https://docs.px4.io/main/en/dev\_setup/building\_px4.html |

## 首次应用教程（Hello Sky）

1. 先决条件

【提示：如果能够完成之前的安装步骤，此先决条件以满足，可以跳过】

您将需要以下内容：

* [PX4 SITL 模拟器](https://docs.px4.io/main/en/simulation/) *或*PX4[兼容的飞行控制器](https://docs.px4.io/main/en/flight_controller/)。
* 适用于所需目标的[PX4 开发工具链。](https://docs.px4.io/main/en/dev_setup/dev_env.html)
* 从 Github[下载 PX4 源代码](https://docs.px4.io/main/en/dev_setup/building_px4.html#download-the-px4-source-code)

PX4-Autopilot源代码[/src/examples/px4\_simple\_app](https://github.com/PX4/PX4-Autopilot/tree/main/src/examples/px4_simple_app)目录包含本教程的完整版本，如果您遇到困难，可以查看该版本。

* 重命名（或删除）**px4\_simple\_app**目录。

1. 最小应用构建步骤

在本节中，我们创建一个仅打印的最小应用程序Hello Sky!。它由一个*C*文件和一个*cmake*定义（告诉工具链如何构建应用程序）组成。如下是完整的构建步骤：

**Step1：**创建一个新目录PX4-Autopilot/src/examples/px4\_simple\_app。

**Step2：**在该目录中创建一个名为px4\_simple\_app.c的新 C 文件：

将如下默认标题复制到页面顶部。这应该出现在所有贡献的文件中！

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* Copyright (c) 2012-2022 PX4 Development Team. All rights reserved.  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* 1. Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \* 2. Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in  \* the documentation and/or other materials provided with the  \* distribution.  \* 3. Neither the name PX4 nor the names of its contributors may be  \* used to endorse or promote products derived from this software  \* without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  \* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  \* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  \* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  \* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  \* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED  \* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  \* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  \* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  \* POSSIBILITY OF SUCH DAMAGE.  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

将以下代码复制到默认标头下方。这应该出现在所有贡献的文件中！

|  |
| --- |
| /\*\*  \* @file px4\_simple\_app.c  \* Minimal application example for PX4 autopilot  \*  \* @author Example User <mail@example.com>  \*/  #include <px4\_platform\_common/log.h>  \_\_EXPORT int px4\_simple\_app\_main(int argc, char \*argv[]);  int px4\_simple\_app\_main(int argc, char \*argv[])  {  PX4\_INFO("Hello Sky!");  return OK;  } |

【提示：主函数必须命名<module\_name>\_main并从模块中导出，如上面代码所示。】

【提示：PX4\_INFO相当于printf（PX4 shell内）（包含在px4\_platform\_common/log.h中）。有不同的日志级别：PX4\_INFO、PX4\_WARN、PX4\_ERR、PX4\_DEBUG。警告和错误会另外添加到[ULog](https://docs.px4.io/main/en/dev_log/ulog_file_format.html)中并显示在[Flight Review上](https://logs.px4.io/)。】

**Step3：**创建并打开一个名为CMakeLists.txt的新*cmake*定义文件。复制以下文本：

|  |
| --- |
| ############################################################################  #  # Copyright (c) 2015 PX4 Development Team. All rights reserved.  #  # Redistribution and use in source and binary forms, with or without  # modification, are permitted provided that the following conditions  # are met:  #  # 1. Redistributions of source code must retain the above copyright  # notice, this list of conditions and the following disclaimer.  # 2. Redistributions in binary form must reproduce the above copyright  # notice, this list of conditions and the following disclaimer in  # the documentation and/or other materials provided with the  # distribution.  # 3. Neither the name PX4 nor the names of its contributors may be  # used to endorse or promote products derived from this software  # without specific prior written permission.  #  # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  # "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  # LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  # FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  # COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  # INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  # BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  # OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED  # AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  # LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  # ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  # POSSIBILITY OF SUCH DAMAGE.  #  ############################################################################  px4\_add\_module(  MODULE examples\_\_px4\_simple\_app  MAIN px4\_simple\_app  STACK\_MAIN 2000  SRCS  px4\_simple\_app.c  DEPENDS  ) |

该px4\_add\_module()方法根据模块描述构建静态库。

* 该MODULE块是模块的固件唯一名称（按照惯例，模块名称的父目录前缀为src）。
* 该MAIN块列出了模块的入口点，该入口点向 NuttX 注册命令，以便可以从 PX4 shell 或 SITL 控制台调用它。

【提示：该px4\_add\_module()格式记录在[PX4-Autopilot/cmake/px4\_add\_module.cmake中](https://github.com/PX4/PX4-Autopilot/blob/main/cmake/px4_add_module.cmake)。】

【笔记：如果您指定DYNAMIC为 的选项px4\_add\_module，则会在 POSIX 平台上创建*共享库*而不是静态库（无需重新编译 PX4 即可加载这些库，并作为二进制文件而不是源代码共享给其他人）。您的应用程序不会成为内置命令，但最终会出现在一个名为examples\_\_px4\_simple\_app.px4mod. 然后，您可以使用以下命令在运行时加载文件来运行命令dyn：dyn ./examples\_\_px4\_simple\_app.px4mod】

**Step4：**创建并打开一个名为Kconfig的新*Kconfig*定义文件，并定义命名符号（请参阅[Kconfig 命名约定](https://docs.px4.io/main/en/hardware/porting_guide_config.html#px4-kconfig-symbol-naming-convention)）。复制以下文本：

|  |
| --- |
| menuconfig EXAMPLES\_PX4\_SIMPLE\_APP  bool “px4\_simple\_app”  default n  ---help---  Enable support for px4\_simple\_app |

**Step5：**构建应用程序/固件：申请现已完成。为了运行它，您首先需要确保它是作为 PX4 的一部分构建的。*应用程序将添加到适合您的目标的板级px4board*文件中的构建/固件中：

* PX4 SITL（模拟器）：[PX4-Autopilot/boards/px4/sitl/default.px4board](https://github.com/PX4/PX4-Autopilot/blob/main/boards/px4/sitl/default.px4board)
* Pixhawk v1/2：[PX4-Autopilot/boards/px4/fmu-v2/default.px4board](https://github.com/PX4/PX4-Autopilot/blob/main/boards/px4/fmu-v2/default.px4board)
* Pixracer (px4/fmu-v4): [PX4-Autopilot/boards/px4/fmu-v4/default.px4board](https://github.com/PX4/PX4-Autopilot/blob/main/boards/px4/fmu-v4/default.px4board)
* 其他板的px4board文件可以在[PX4-Autopilot/boards/](https://github.com/PX4/PX4-Autopilot/tree/main/boards)中找到

要将应用程序编译到固件中，请把以下代码添加到px4board类型文件中最后一行：

|  |
| --- |
| CONFIG\_EXAMPLES\_PX4\_SIMPLE\_APP=y |

目的是在*px4board*文件中添加相应的 Kconfig 键。或者运行[boardconfig](https://docs.px4.io/main/en/hardware/porting_guide_config.html#px4-menuconfig-setup) make px4\_fmu-v4\_default boardconfig（在*px4board*文件中添加相应的 Kconfig 键【提示：不推荐使用，主要是我也不会操作这个界面】）：

|  |
| --- |
| examples --->  [x] PX4 Simple app ---- |

**Step6：**测试应用程序（硬件）【提示：最开始的用的时候不建议去弄硬件测试，后面有仿真软件测试部分】

**Step6.1：**将固件上传到您的主板

启用上传器，然后重置开发板：

* Pixhawk v1/2：make px4\_fmu-v2\_default upload
* 皮克斯霍克 v3：make px4\_fmu-v4\_default upload

在重置板之前，它应该打印一些编译消息，并在最后打印：

|  |
| --- |
| Loaded firmware for X,X, waiting for the bootloader... |

一旦板重置并上传，它就会打印：

|  |
| --- |
| Erase : [====================] 100.0%  Program: [====================] 100.0%  Verify : [====================] 100.0%  Rebooting.  [100%] Built target upload |

**Step6.2：**连接控制台

现在通过串行或 USB连接到[系统控制台。](https://docs.px4.io/main/en/debug/system_console.html)按**ENTER**键将弹出 shell 提示符：

|  |
| --- |
| nsh> |

输入“help”并按 ENTER 键，它就会打印如下结果：

|  |
| --- |
| nsh> help  help usage: help [-v] [<cmd>]  [ df kill mkfifo ps sleep  ? echo losetup mkrd pwd test  cat exec ls mh rm umount  cd exit mb mount rmdir unset  cp free mkdir mv set usleep  dd help mkfatfs mw sh xd  Builtin Apps:  reboot  perf  top  ..  px4\_simple\_app  ..  sercon  serdis |

请注意，它px4\_simple\_app现在是可用命令的一部分。通过键入px4\_simple\_app并 ENTER 启动它，将得到如下结果：

|  |
| --- |
| nsh> px4\_simple\_app  Hello Sky! |

该应用程序现在已在系统中正确注册，并且可以扩展以实际执行有用的任务。

**Step7：**测试应用程序（SITL）【提示：仿真软件测试部分】

如果您使用 SITL，则*PX4 控制台*会自动启动（请参阅[构建代码 > 首次构建（使用 jMAVSim 模拟器）](https://docs.px4.io/main/en/dev_setup/building_px4.html#first-build-using-the-jmavsim-simulator)）。与*nsh 控制台*（请参阅上一节）一样，您可以键入help以查看内置应用程序的列表。

输入px4\_simple\_app以运行最小应用程序，将得到如下结果：

|  |
| --- |
| pxh> px4\_simple\_app  INFO [px4\_simple\_app] Hello Sky! |

现在可以扩展该应用程序以实际执行有用的任务。

## 订阅传感器数据

为了做一些有用的事情，应用程序需要订阅输入并发布输出（例如电机或伺服命令）。

【提示：请先完成最小应用程序px4\_simple\_app，然后在此基础上修改px4\_simple\_app.c中的代码（或者不修改，原先的代码就是这节所用到的代码），完成该节内容，】

【提示：PX4 硬件抽象的优势在这里发挥出来！无需以任何方式与传感器驱动程序交互，如果板或传感器更新，也无需更新您的应用程序。】

应用程序之间的各个消息通道称为[topics](https://docs.px4.io/main/en/middleware/uorb.html)（主题）。[在本教程中，我们对topics中的SensorCombined](https://github.com/PX4/PX4-Autopilot/blob/main/msg/SensorCombined.msg)感兴趣 ，它保存整个系统的同步传感器数据。

**Step1：**订阅主题

订阅主题很简单，如下代码所示：

|  |
| --- |
| #include <uORB/topics/sensor\_combined.h>  ..  int sensor\_sub\_fd = orb\_subscribe(ORB\_ID(sensor\_combined)); |

这sensor\_sub\_fd是一个主题句柄，可用于非常有效地执行对新数据的阻塞等待。当前线程进入睡眠状态，一旦有新数据可用，就会被调度程序自动唤醒，等待时不会消耗任何 CPU 周期。为此，我们使用POSIX 系统调用[poll()](http://pubs.opengroup.org/onlinepubs/007908799/xsh/poll.html)。

添加poll()到订阅看起来像（*伪代码，在下面查找完整的实现*）：

|  |
| --- |
| #include <poll.h>  #include <uORB/topics/sensor\_combined.h>  ..  int sensor\_sub\_fd = orb\_subscribe(ORB\_ID(sensor\_combined));  /\* one could wait for multiple topics with this technique, just using one here \*/  px4\_pollfd\_struct\_t fds[] = {  { .fd = sensor\_sub\_fd, .events = POLLIN },  };  while (true) {  /\* wait for sensor update of 1 file descriptor for 1000 ms (1 second) \*/  int poll\_ret = px4\_poll(fds, 1, 1000);  ..  if (fds[0].revents & POLLIN) {  /\* obtained data for the first file descriptor \*/  struct sensor\_combined\_s raw;  /\* copy sensors raw data into local buffer \*/  orb\_copy(ORB\_ID(sensor\_combined), sensor\_sub\_fd, &raw);  PX4\_INFO("Accelerometer:\t%8.4f\t%8.4f\t%8.4f",  (double)raw.accelerometer\_m\_s2[0],  (double)raw.accelerometer\_m\_s2[1],  (double)raw.accelerometer\_m\_s2[2]);  }  } |

输入以下内容再次编译应用程序：

|  |
| --- |
| make |

**Step2：**测试uORB订阅

最后一步是通过在 nsh shell 中键入以下内容将应用程序作为后台进程/任务启动：

|  |
| --- |
| px4\_simple\_app & |

您的应用程序将在控制台中显示 5 个传感器值，然后退出：

|  |
| --- |
| [px4\_simple\_app] Accelerometer: 0.0483 0.0821 0.0332  [px4\_simple\_app] Accelerometer: 0.0486 0.0820 0.0336  [px4\_simple\_app] Accelerometer: 0.0487 0.0819 0.0327  [px4\_simple\_app] Accelerometer: 0.0482 0.0818 0.0323  [px4\_simple\_app] Accelerometer: 0.0482 0.0827 0.0331  [px4\_simple\_app] Accelerometer: 0.0489 0.0804 0.0328 |

【提示：[完整应用程序的模块模板](https://docs.px4.io/main/en/modules/module_template.html)可用于编写可从命令行控制的后台进程。】

**Step3：**发布数据

要使用计算的输出，下一步是发布结果。下面我们展示如何发布attitude主题。

【笔记：我们之所以选择attitude主题，是因为我们知道mavlink应用程序会将其转发到地面控制站，这提供了一种查看结果的简单方法。】

界面非常简单：初始化要发布的主题数据结构体，并宣传该主题：

|  |
| --- |
| #include <uORB/topics/vehicle\_attitude.h>  ..  /\* advertise attitude topic \*/  struct vehicle\_attitude\_s att;  memset(&att, 0, sizeof(att));  orb\_advert\_t att\_pub\_fd = orb\_advertise(ORB\_ID(vehicle\_attitude), &att); |

在主循环中，只要准备好就发布信息：

|  |
| --- |
| orb\_publish(ORB\_ID(vehicle\_attitude), att\_pub\_fd, &att); |

**Step4：**运行完整示例

最后使用下面代码运行你的应用程序：

|  |
| --- |
| px4\_simple\_app |

如果启动QGroundControl，您可以检查实时图中的传感器值([Analyze > MAVLink Inspector](https://docs.qgroundcontrol.com/master/en/analyze_view/mavlink_inspector.html))。【提示：与此程序无关，QGroundControl自带有的功能】

**附录**（完整的示例代码）

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* Copyright (c) 2012-2019 PX4 Development Team. All rights reserved.  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* 1. Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \* 2. Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in  \* the documentation and/or other materials provided with the  \* distribution.  \* 3. Neither the name PX4 nor the names of its contributors may be  \* used to endorse or promote products derived from this software  \* without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  \* FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  \* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  \* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  \* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  \* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED  \* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  \* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  \* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  \* POSSIBILITY OF SUCH DAMAGE.  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* @file px4\_simple\_app.c  \* Minimal application example for PX4 autopilot  \*  \* @author Example User <mail@example.com>  \*/  #include <px4\_platform\_common/px4\_config.h>  #include <px4\_platform\_common/tasks.h>  #include <px4\_platform\_common/posix.h>  #include <unistd.h>  #include <stdio.h>  #include <poll.h>  #include <string.h>  #include <math.h>  #include <uORB/uORB.h>  #include <uORB/topics/sensor\_combined.h>  #include <uORB/topics/vehicle\_attitude.h>  \_\_EXPORT int px4\_simple\_app\_main(int argc, char \*argv[]);  int px4\_simple\_app\_main(int argc, char \*argv[])  {  PX4\_INFO("Hello Sky!");  /\* subscribe to sensor\_combined topic \*/  int sensor\_sub\_fd = orb\_subscribe(ORB\_ID(sensor\_combined));  /\* limit the update rate to 5 Hz \*/  orb\_set\_interval(sensor\_sub\_fd, 200);  /\* advertise attitude topic \*/  struct vehicle\_attitude\_s att;  memset(&att, 0, sizeof(att));  orb\_advert\_t att\_pub = orb\_advertise(ORB\_ID(vehicle\_attitude), &att);  /\* one could wait for multiple topics with this technique, just using one here \*/  px4\_pollfd\_struct\_t fds[] = {  { .fd = sensor\_sub\_fd, .events = POLLIN },  /\* there could be more file descriptors here, in the form like:  \* { .fd = other\_sub\_fd, .events = POLLIN },  \*/  };  int error\_counter = 0;  for (int i = 0; i < 5; i++) {  /\* wait for sensor update of 1 file descriptor for 1000 ms (1 second) \*/  int poll\_ret = px4\_poll(fds, 1, 1000);  /\* handle the poll result \*/  if (poll\_ret == 0) {  /\* this means none of our providers is giving us data \*/  PX4\_ERR("Got no data within a second");  } else if (poll\_ret < 0) {  /\* this is seriously bad - should be an emergency \*/  if (error\_counter < 10 || error\_counter % 50 == 0) {  /\* use a counter to prevent flooding (and slowing us down) \*/  PX4\_ERR("ERROR return value from poll(): %d", poll\_ret);  }  error\_counter++;  } else {  if (fds[0].revents & POLLIN) {  /\* obtained data for the first file descriptor \*/  struct sensor\_combined\_s raw;  /\* copy sensors raw data into local buffer \*/  orb\_copy(ORB\_ID(sensor\_combined), sensor\_sub\_fd, &raw);  PX4\_INFO("Accelerometer:\t%8.4f\t%8.4f\t%8.4f",  (double)raw.accelerometer\_m\_s2[0],  (double)raw.accelerometer\_m\_s2[1],  (double)raw.accelerometer\_m\_s2[2]);  /\* set att and publish this information for other apps  the following does not have any meaning, it's just an example  \*/  att.q[0] = raw.accelerometer\_m\_s2[0];  att.q[1] = raw.accelerometer\_m\_s2[1];  att.q[2] = raw.accelerometer\_m\_s2[2];  orb\_publish(ORB\_ID(vehicle\_attitude), att\_pub, &att);  }  /\* there could be more file descriptors here, in the form like:  \* if (fds[1..n].revents & POLLIN) {}  \*/  }  }  PX4\_INFO("exiting");  return 0;  } |

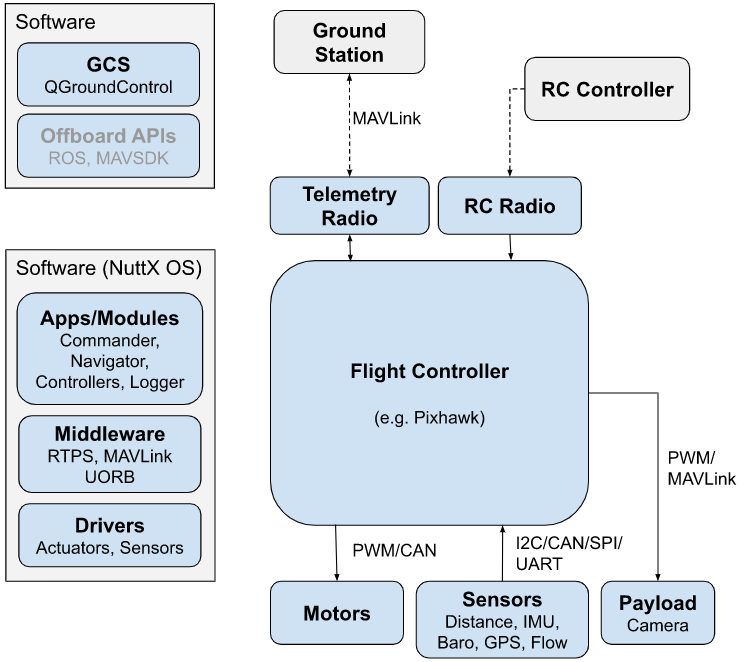
## PX4系统架构

以下部分提供了两个“典型”PX4 系统的 PX4 硬件和软件堆栈的高级概述；一个只有一个飞行控制器，另一个有一个飞行控制器和一台配套计算机（也称为“任务计算机”）。

【笔记：PX4架构[概述](https://docs.px4.io/main/en/concept/architecture.html)提供了有关飞行堆栈和中间件的信息。[ROS](https://docs.px4.io/main/en/ros/)和[MAVSDK](https://mavsdk.mavlink.io/main/en/)涵盖了 Offboard API 。】

**Part1：**飞行控制器（仅限）

下图提供了基于飞行控制器的典型“简单”PX4 系统的高级概述。



硬件包括：

* [飞行控制器](https://docs.px4.io/main/en/flight_controller/)（运行 PX4 飞行堆栈）。这通常包括内部 IMU、指南针和气压计。
* [Motor ESCs](https://docs.px4.io/main/en/peripherals/esc_motors.html) 连接到[PWM 输出](https://docs.px4.io/main/en/peripherals/pwm_escs_and_servo.html)、[DroneCAN](https://docs.px4.io/main/en/dronecan/escs.html)（DroneCAN 允许双向通信，而不是如图所示的单向通信）或其他总线的
* 通过 I2C、SPI、CAN、UART 等连接的传感器（[GPS](https://docs.px4.io/main/en/gps_compass/)、[指南针](https://docs.px4.io/main/en/gps_compass/)、距离传感器、气压计、光流、气压计、ADSB 应答器等）。
* [相机](https://docs.px4.io/main/en/peripherals/camera.html)或其他有效负载。相机可以连接到 PWM 输出或通过 MAVLink进行连接。
* 用于连接地面站计算机/软件的遥测无线电[Telemetry radios](https://docs.px4.io/main/en/telemetry/)。
* 用于手动控制的RC控制系统[RC Control System](https://docs.px4.io/main/en/getting_started/rc_transmitter_receiver.html)。

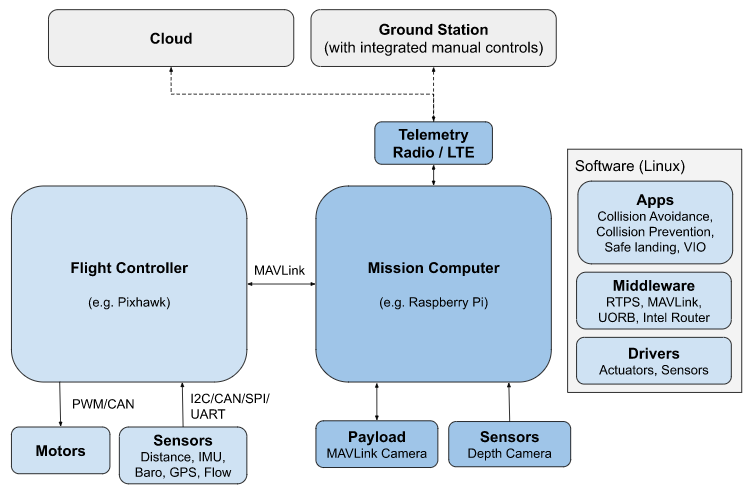
软件包括：

上图的左侧显示了软件堆栈，它与该图的硬件部分（大约）水平对齐。

* 地面站计算机通常运行[QGroundControl](https://docs.px4.io/main/en/getting_started/px4_basic_concepts.html#qgroundcontrol)（或一些其他地面站软件）。它还可以运行[MAVSDK](https://mavsdk.mavlink.io/)等机器人软件或 [ROS](https://docs.px4.io/main/en/ros/)。
* 飞行控制器上运行的 PX4 飞行堆栈包括[驱动程序](https://docs.px4.io/main/en/modules/modules_driver.html)、[通信模块](https://docs.px4.io/main/en/modules/modules_communication.html)、[控制器](https://docs.px4.io/main/en/modules/modules_controller.html)、[估算器](https://docs.px4.io/main/en/modules/modules_controller.html)以及其他[中间件和系统模块](https://docs.px4.io/main/en/modules/modules_main.html)。

**Part2：**飞行控制器和配套计算机

下图显示了一个 PX4 系统，其中包括飞行控制器和配套计算机（此处称为“任务计算机”）。



飞行控制器运行正常的 PX4 飞行堆栈，而配套计算机则提供[物体回避](https://docs.px4.io/main/en/computer_vision/obstacle_avoidance.html)和[碰撞预防](https://docs.px4.io/main/en/computer_vision/collision_prevention.html)等高级功能。这两个系统使用快速串行或 IP 链路连接，通常使用[MAVLink 协议进行通信](https://mavlink.io/en/)。与地面站和云端的通信通常通过配套计算机进行路由（例如使用[MAVLink 路由器](https://github.com/mavlink-router/mavlink-router)（来自英特尔））。

PX4 系统通常在配套计算机上运行 Linux 操作系统（因为[PX4/PX4-Avoidance](https://github.com/PX4/PX4-Avoidance)项目提供了专为 Linux 设计的基于 ROS 的回避库）。Linux 是一个比 NuttX 更好的“通用”软件开发平台；还有更多的 Linux 开发人员，并且已经编写了许多有用的软件（例如计算机视觉、通信、云集成、硬件驱动程序）。出于同样的原因，配套计算机有时也会运行 Android。

【笔记：该图显示了通过 LTE 进行的云或地面站连接，这种方法已在许多基于 PX4 的系统中使用。PX4 不提供专门用于 LTE 和/或云集成的软件（这需要定制开发）。】

## PX4飞行堆栈架构

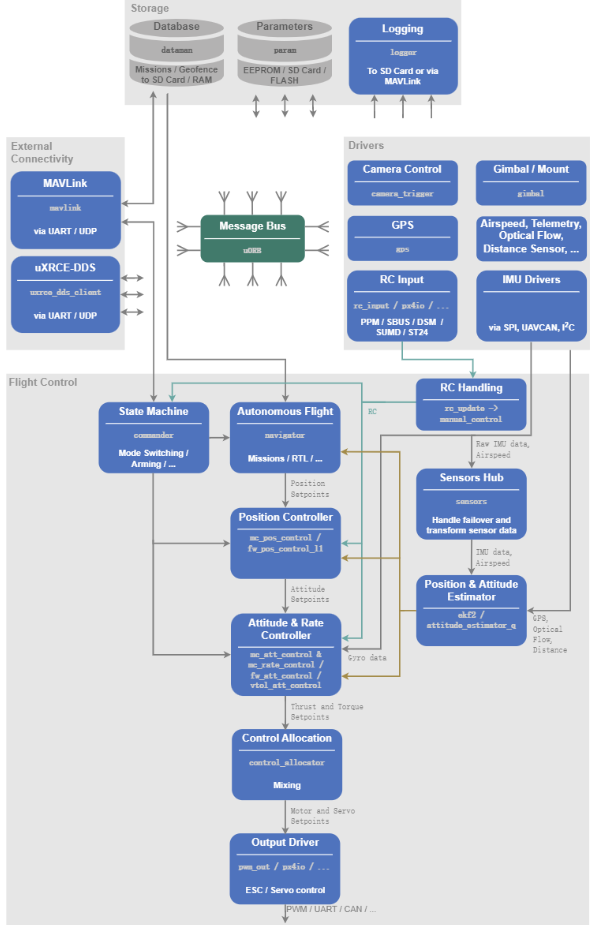
PX4由两个主要层组成：[飞行堆栈](https://docs.px4.io/main/en/concept/architecture.html#flight-stack)是一个估计器和飞行控制系统，[中间件](https://docs.px4.io/main/en/concept/architecture.html#middleware)是通用机器人层，可以支持任何类型的自主机器人，提供内部/外部通信和硬件集成。

所有 PX4 [airframes](https://docs.px4.io/main/en/airframes/)（机身）共享一个代码库（这包括其他机器人系统，如船只、漫游车、潜艇等）。整个系统设计是[反应式的](http://www.reactivemanifesto.org/)， 意思就是：

* 所有功能都分为可交换和可重用的组件
* 通信是通过异步消息传递完成的
* 系统可以处理不同的工作负载

1. 高级软件架构

下图详细概述了 PX4 的构建模块。该图的顶部包含中间件块，而下部显示飞行堆栈的组件。



1. 飞行堆栈
2. 中间件
3. 更新率
4. 运行环境
5. 后台任务
6. 操作系统特定信息