DJI Onboard API 说明文档

V1.0.0

2015.06

目录

DJI 为开发者提供两种功能完善的飞行控制 API 帮助开发飞行应用：Mobile API 和 Onboard API。Mobile API 是 DJI Mobile SDK 的核心部分，开发者可以基于 iOS/Android 系统上的 SDK 库编写控制飞行器的移动端应用。而 Onboard API 则提供串行接口（UART），允许开发者将自己的计算设备挂载到飞行器上，通过有线的方式直接控制飞行器。

本文档介绍 Onboard API。文档包括两个部分：第一部分是快速入门，包括一些主要功能的介绍；第二部分是编程指南，介绍所有进行程序控制所需的知识。

快速入门

在快速入门中，我们首先对 API 做一个整体的介绍，并且定义一些关键的术语。然后我们用一份示例代码介绍使用 Onboard API 的主要步骤。

概述

DJI Matrice 100 开发者飞行器（以下简称 MATRICE 100）是一个为二次开发专门设计的四轴飞行器。这款飞行器有宽阔的甲板可用于安装机载设备。它的可拆卸电池仓、扩展设备安装架、额外的电源接口都允许用户设计紧凑、功能强大的飞行器应用。Onboard API 则是为了辅助 MATRICE 100 的使用而设计的串口 API。

Onboard API 的主要特性

- 可靠的通讯协议
  基于会话的通信可以防止数据丢包。32bit 的 CRC 进一步降低通信异常的可能性。
- 灵活的控制方式
  可对飞行器输入多种控制量，包括位置、速度和姿态控制
- 可配置的外发数据
  飞控外发的数据可以进行种类和频率的配置，节省带宽
- 考虑自动飞行的设计
  飞行状态控制指令和外发指令的设计都考虑到自动导航应用的实际需求

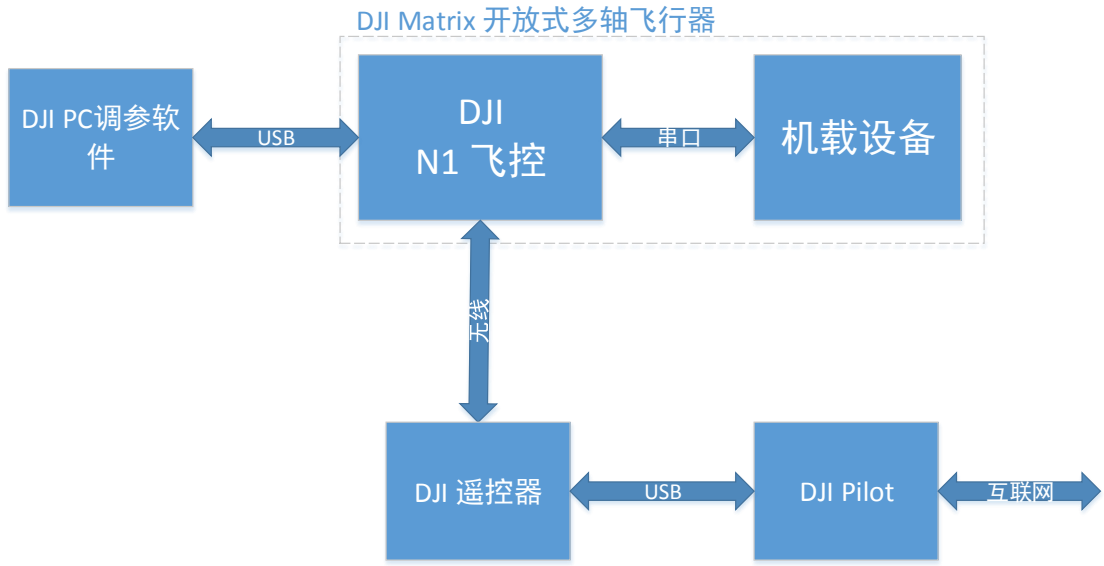
系统综述

整个系统的核心设备是 MATRICE 100 以及装载在 MATRICE 100 上的机载设备。机载设备与 MATRICE 100 的飞控（N1 飞控）通过串口线连接。机载设备可以是任何能够进行串口通信和 AES 加解密的计算设备。

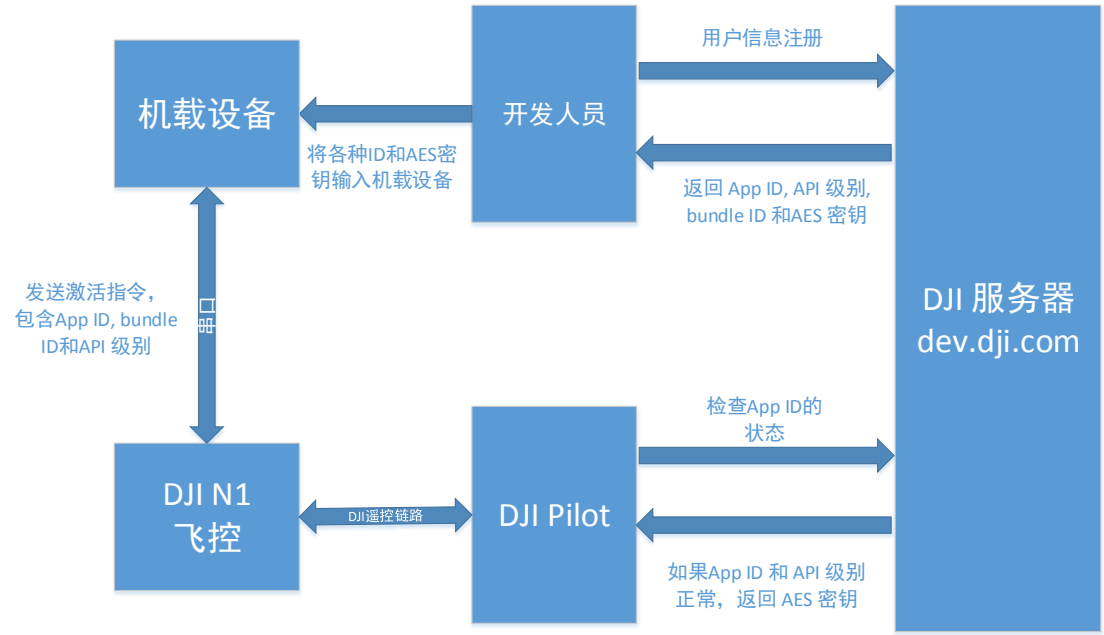
DJI N1 PC 调参软件用于配置 MATRICE 100 的串口和 MATRICE 100 固件的升级。这款调参软件与 DJI 的其他 PC 软件使用方法相同。由于新一代的 DJI 产品主要通过手机软件 DJI Pilot 进行配置，PC 调参软件只保留了一些无法通过手机软件实现的功能，比如固件升级和串口配置。

根据相关的法律法规，由于 Onboard API 可以让用户设计超视距飞行的自动系统，DJI 必须对 MATRICE 100 采用更严格的监管制度。在使用 MATRICE 100 之前，开发者必须在 *dev.dji.com* 网站（以下简称 DJI 服务器）上进行注册，然后激活飞行器。DJI 服务器会为开发者生成 App ID 和 AES 密钥。机载设备和 MATRICE 100 之间的大部分通信必须进行加密，这个密钥会在激活过程中由 DJI 服务器发给机载设备。激活和加密过程会在“编程指南-命令集 0x00”中详细介绍。

系统架构框图：



注册和激活的过程图：



激活过程中的一个重要概念是设备许可数量（DAN）。它有以下性质：

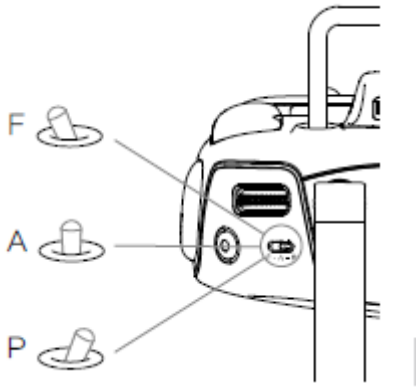
- 每个 App ID 对应一个 DAN。它表示着这个 App ID 可以激活的飞控数量。
- 每个新的 App ID 的 DAN 上限默认为 5。
- 在激活过程中，每一个新的飞控都会让 App ID 的 DAN 增加 1。如果某个 App ID 的 DAN

达到上限，那么激活就会失败，新的飞控不能激活。

- 开发者可以在 dev.dji.com 上申请增加 DAN 的上限。

遥控器、Onboard API 和 Mobile API

MATRICE 100 被设计为可以使用遥控器、机载设备和移动设备进行控制。为 Inspire 1 和 Phantom3 系列设计的标准 DJI 辅助软件“DJI Pilot”也可以在 MATRICE 100 上使用。另外 DJI Mobile SDK 也适用于 MATRICE 100（请访问 *dev.dji.com* 了解关于 DJI Mobile SDK 的更多信息）。因为 MATRICE 100 有三种可能的控制输入，控制输入必须有优先级管理。



我们将遥控器设计为控制权限最高的设备。遥控器可以选择是否将控制权转给 API 控制。如果遥控器切换到 API 控制模式（在调参软件中配置 MATRICE 100，然后将遥控器模式选择开关切到 F 档，请见右侧示意图），设备通过 Onboard API 和 Mobile API 可以请求获得控制权。移动设备的 API 有更高的控制优先级。如果移动设备先获得了控制权，那么机载设备无法成功获得控制权。

本文档着重介绍 Onboard API。我们假设开发者在使用 Onboard API 的过程中不使用 mobile API。在目前版本中，混合控制（同时使用 Mobile API 和 Onboard API）没有得到完整支持。

命令权限等级

开发者在 *dev.dji.com* 上注册的时候，DJI 服务器会为开发者分配一个 API 命令权限级别，这个级别根据开发者的需求和开发能力决定。开发者在使用机载设备的时候，必须将这个权限级别存入机载设备的程序中用于激活。MATRICE 100 的飞控会在激活过程中检查权限级别的有效性。

不同的 API 级别对应着不同级别的飞行控制命令。

- Level 0. 激活相关的命令
- Level 1. 数据读取和非飞行控制命令。包括相机和云台的控制、飞行数据监测、传感器数据读取。这个级别不包含对飞行器的运动控制。
- Level 2. 飞行控制。包括对飞行器的运动控制，而且可以控制飞行器的飞行状态，比如降落、返航等等。

只有富有经验的开发者才能够使用权限级别高于 2 的飞行控制命令。在未来的 onboard API 中，我们会不断增加各种权限级别的控制命令。

[ROS]通过无线串口控制 MATRICE 100

在这个示例中，我们使用示例代码“dji_sdk_keyboard_ctrl”对 MATRICE 100 进行远程控制。该代码基于 ROS 软件包 keyboardteleop.js。我们设计了一个简单的 HTML GUI 帮助开发者熟悉如何使用键盘和鼠标控制 MATRICE 100。

必备硬件

1. MATRICE 100 多轴飞行器
2. DJI 串口连接线
   （包含在 MATRICE 100 附件当中）
3. 若干杜邦线
4. 一对配置好的 230400 无线串口
   （淘宝购买）
5. USB-TTL 转换接头
   （淘宝购买）
   注意：Windows/Mac 上使用 USB-TTL 转换接头需要安装 PL2303 驱动
6. 5V 输出 DC-DC 电源模块
   （淘宝购买）
   注意：MATRICE 100 上不提供 5V 的电源输出，所以为了给无线串口供电，开发者需要自己购买 5V 电源模块

必备软件

1. 装有 DJI N1 PC 调参软件的 windows 电脑
2. 装有 DJI Pilot（最新版本）的可联网移动设备
3. 装有 Ubuntu 14.04 (或更高版本) 和 ROS Indigo（或更高版本）的 Linux 电脑。示例代码在 ROS Indigo 上进行过测试。
4. ROS package rosbridge_server
5. 示例代码 dji_sdk 与 dji_keyboard_ctrl

步骤

- MATRICE 100 飞行器配置。给飞控上电，DJI N1 PC 调参软件可以用于升级固件和配置启用 API 模式。
  在页面“基础”当中，开发者可以勾选“启用 API 控制”来配置遥控器和 MATRICE 100 启用 API 模式控制相关的功能。开发者能够通过“串口波特率和外发数据设置”区的选项来配置串口波特率和飞控外发数据的内容。

启用 API 控制之后，将遥控器模式开关置为中位（F 档）。

- 连接无线串口。电脑安装 USB-TTL 软件驱动，通过 USB-TTL 连接一个串口模块。另一个串口模块连接飞控附带的串口转接线。注意放在 MATRICE 100 上的串口模块需要 5V 电源模块单独供电。5V 电源模块可以使用 MATRICE 100 的电源作为输入。
- 启动示例代码.

1. 编译 ROS package dji_sdk
2. 启动 roscore，然后启动 rosbridge_server

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

3. 通过代码中的 launch 文件启动 dji_sdk_node。

下面的代码是示例 launch 文件

```
<launch>
    <node pkg="dji_sdk" type="dji_sdk_node" name="dji_sdk_node" output="screen">
        <!-- node parameters -->
        <param name="serial_name" type="string" value="/dev/ttySAC0"/>
        <param name="baud_rate" type="int" value="230400"/>
        <param name="app_id" type="int" value="<!-- your appid -->"/>
        <param name="app_api_level" type="int" value="<!-- your app level -->"/>
        <param name="app_version" type="int" value="<!-- your app version -->"/>
        <param name="app_bundle_id" type="string" value="<!-- your app bundle id
-->"/>
        <param name="enc_key" type="string" value="<!-- your app enc key -->"/>

    </node>
```

```
</launch>
```

其中的 node parameters 含义如下

Name	Type	说明
serial_name	String	串口设备名。通常为“/dev/ttyUSB0”，但是在不同的 Linux 设备上可能有不同的名称。 “ls /dev/*”和”dmesg | tail”命令可以用于查询设备名。
baud_rate	Int	串口波特率，必须与通过调参软件设置的相同。
app_id	Int	*dev.dji.com* 服务器返回的 AppID
app_api_level	Int	*dev.dji.com* 服务器返回的 API 级别
app_version	Int	开发者设定的应用版本号
app_bundle_id	String	*dev.dji.com* 服务器返回的 bundle ID
enc_key	string	*dev.dji.com* 服务器返回的 AES 密钥

注意：这条命令一定要在 sudo su 模式下启动，因为打开串口需要 root 权限。

```
sudo su

roslaunch dji_sdk sdk_demo.launch
```

4. 编辑“sdk_keyboard_demo.html”，把 url 中的地址改成 Linux 系统的主机名或者 localhost（127.0.0.1）

```
function init() {
    // Connecting to ROS.
    var ros = new ROSLIB.Ros({
        url: 'ws://127.0.0.1:9090'
    });
}
```

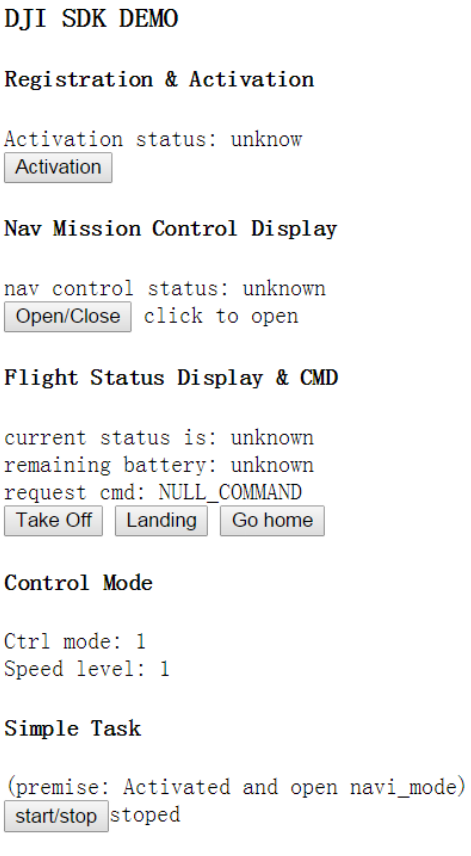
5. 在浏览器中打开“sdk_keyboard_demo.html”。rosbridge_server 会显示有新的 client 连接上，否则请检查步骤 4 中的设置。此时可以在页面中读取到飞行平台的状态信息。

- 测试通信链路。在 sdk_keyboard_demo 页面中点击“Activation”。如果 PC 和飞行平台之间的链接畅通，页面上可以看到返回码。否则请检查链路连接状况。

- 激活飞行平台。连接装有 DJI Pilot 的手机和飞行平台的遥控器，确保手机连接到网络。激活过程会在点击“Activation”之后自动完成。

- 启动飞行器在空中悬停，与周围物体保持安全距离。遥控器切入 F 档，通过程序请求控制权，此时可以通过示例程序发送控制指令。除了下图所示的指令外，键盘“WASD”键控制飞行器向对应方向的倾角，“ZC” 控制竖直速度、“QE”控制偏航旋转。“WASD”控制的倾角度数为 5*speed_level。这个 speed_level 默认为 1，可以通过键盘数字键 123456 来修改。speed_level 修改后，姿态控制指令的数值也会随之改变。请谨慎使用大姿态角

度的指令，飞行器会很快加速。



- 安全飞行注意事项：配置完成后用户必须把遥控器切到 F 档 API 控制模式，飞控才会准备接受串口控制。此后机载设备可以发送请求获得控制权。这样的设计是将遥控器的模式切换拨杆作为紧急情况下的保险开关。任何情况下用户拨动遥控器模式切换拨杆，都会退出 API 控制模式。我们建议两位开发者一起合作进行测试，一个人控制网页 GUI，另一个人使用遥控器紧急制动。

  再次进入 API 控制模式之后，机载设备不需要再次请求获得控制权。

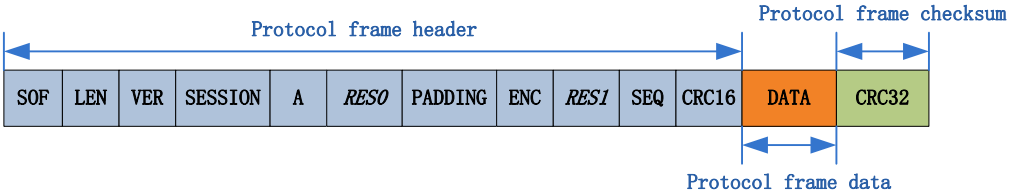
  如果遥控器和飞控上电时遥控器已经置于 F 档，则需要切出再切入 F 档一次才可触发 API 控制模式，这样可以防止 API 控制模式在用户没有意识到的情况下被打开。

编程指南

编程指南中介绍了如何通过程序与 MATRICE 100 交互并发送控制指令。我们推荐开发者先通过快速入门实现我们的示例代码，然后再阅读编程指南。

1 协议说明

1.1 协议格式



1.2 协议格式说明

字段	字节索引	大小（单位 bit）	说明
SOF	0	8	帧起始标识。固定位 0xAA
LEN	1	10	帧长度。最大为 1023 bytes
VER		6	协议版本
SESSION	3	5	通信过程中的会话 ID
A		1	帧标识 0：数据帧 1：应答帧
RES0		2	保留不用。固定值为 0x0
PADDING	4	5	加密帧数据时附加的数据长度
ENC		3	帧数据加密类型 0：不加密 1：AES 加密
RES1	5	24	保留不用。固定值为 0x0
SEQ	8	16	帧序列号
CRC16	10	16	帧头 CRC16 校验值
DATA	12	--①	帧数据段。最大长度为 1007bytes
CRC32	--②	32	帧 CRC32 校验值

①：长度大小不固定，最大长度为 1007。

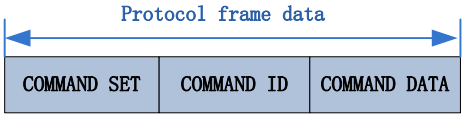
②：字节索引根据 DATA 长度大小而定。

1.3 协议数据段说明

飞控和机载设备通信的数据包分为三类：

一．命令数据包。从机载设备发送到飞控，包含对飞行器的控制指令。
二．信息数据包。从飞控发送到机载设备，包含飞控的各种状态信息和传感器数据。
三．应答数据包。从飞控发送到机载设备，包含控制指令的执行结果。

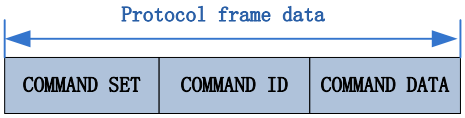
1.3.1 机载设备发送给飞控的命令数据包的数据段格式



字段	字节索引	大小（单位 byte）	说明
COMMAND SET	0	1	命令集
COMMAND ID	1	1	命令码
COMMAND DATA	2	--①	命令数据

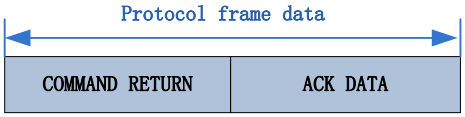
①：大小根据具体命令而定。

1.3.2 飞控发给机载设备的信息数据包的数据段格式



字段	字节索引	大小（单位 byte）	说明
COMMAND SET	0	1	命令集
COMMAND ID	1	1	命令码
COMMAND DATA	2	--	飞机状态及传感器等数据

1.3.3 飞控发给机载设备的应答数据包的数据段格式



字段	字节索引	大小（单位 byte）	说明
COMMAND RETURN	0	2	命令执行的返回信息
ACK DATA	2	--	应答数据

1.4 通信会话机制

协议设计使用了会话机制，以保证命令数据和应答数据不会因为丢包而出现通信双方异常。通信双方在向对方发起通信会话时，可以根据需要通过设置协议的 SESSION 字段来选择会话方式。协议中设计了三种会话方式。

会话方式	SESSION	描述
方式 1	0	发送端不需要接收端应答
方式 2	1	发送端需要接收端应答数据，但是可以容忍应答数据丢包
方式 3	2-31	发送端需要正确收到接收端的应答包。发送端使用这些 session 发送命令数据包时，接收端应答后要保存当前的应答包作为该 session 的应答数据，应答包中包含该命令数据包中的 sequence number 和 session id。如果通信过程中，发送端没有正确收到应答包，可以重新发送该命令数据包，接收端收到后将保存的应答包重新发送回去。下一次，如果发送端使用和上一次相同的 session id，但不同的 sequence number 来发送命令数据包时，接收端会丢弃上一次保存的 session 应答数据，重新保存新的 session 应答数据。

1.5 API 示例

假设使用以下 C/C++枚举类型表示会话方式：

```
enum SESSION_MODE
{
	SESSION_MODE1,
	SESSION_MODE2,
	SESSION_MODE3,
}
```

命令数据包的回调接口函数定义如下：

```
typedef void(*CMD_CALLBACK_FUNC)(const void* p_data, unsigned int n_size)
```

假设通信中发送协议数据的函数定义如下：

```
unsigned int Linklayer_Send(SESSION_MODE session_mode,const void* p_data, unsigned int n_size, char enc_type,unsigned short ack_timeout ,unsigned char retry_time,CMD_CALLBACK_FUNC cmd_callback)
```

参数 session_mode：会话方式。

参数 p_data：指向待发协议数据流的起始地址。

参数 n_size：协议数据流的大小。

参数 enc_type：是否采用加密发送。

参数 ack_timeout：使用会话方式 3 时接收端应答的超时时间，单位 ms。

参数 retry_time:使用会话方式 3 接收端不应答时，发送端重发的次数。

参数 cmd_callback:回调函数接口。

备注：由于会话方式 3 是一种可靠会话方式，开发者在协议链路层实现中应考虑数据丢包后的重发机

2 命令集说明

2.1 命令及权限

DJI Onboard API 相关的命令分为三大类：

类别	说明	命令集代码
激活验证类	该命令集包含的 ID 只与激活相关	0x00
飞控接受的控制	控制数据的命令集	0x01
飞控外发的数据	飞控外发的命令集	0x02

每类命令有唯一的命令集代码，命令集包含的所有命令有各自的命名码和命令数据。
飞控接受的控制命令全部有权限级别。在未来版本中会有更多的控制命令以不同的权限级别开放。目前权限级别如下：

权限级别（API 级别）	权限描述
0	API 激活命令
1	相机和云台的控制命令
2	飞行控制命令

2.2 命令集

2.2.1 命令集 0x00 激活验证类

API 激活验证命令集的所有命令权限级别为 0，即所有用户都可以使用命令集中的命令对飞机进行激活与版本查询等操作。激活 API 通过 DJI Pilot 与 DJI Server 连接，需要手机连接互联网。

2.2.1.1 命令码 0x00 获取 API 版本

	偏移（字节）	*大小（字节）*	*说明*
请求数据	**1**	1	任意请求数据
应答数据	**0**	2	返回码 0x0000：激活成功 0xFF00：命令不支持 0xFF01：机载设备无授权 0xFF02：机载设备权限不足

	偏移（字节）	*大小（字节）*	*说明*
	2	4	CRC32, 字符串版本的 CRC32
	6	--	不定长，最大长度 32bytes。有效部分到'\0'结尾

推荐接收应答数据的 C/C++结构体：

```
typedef struct
{
    unsigned short version_ack;
    unsigned int version_crc;
    signed char version_name[32];
} version_query_data_t;
```

假设获取 API 版本命令的回调函数为：

```
void print_sdk_version(const void* p_data, unsigned int n_size)
{
    version_query_data_t* p_version = (version_query_data_t*)p_data;
    if (p_version->version_ack == 0)
    {
        printf("%s\n", p_version->version_name);
    }
}
```

应用程序中发送请求获取 API 版本命令的操作如下：

```
unsigned char cmd_buf[3];
cmd_buf[0] = 0x00;//command set
cmd_buf[1] = 0x00;//command id
cmd_buf[2] = 0x00;//command data
Linklayer_Send(SESSION_MODE3,
               cmd_buf,
               3,
               0,
               200,
               3,
               print_sdk_version
);
```

以上使用的会话方式 3 进行 API 版本获取请求，飞机收到请求并响应后，应用程序相应的回调函数 print_sdk_version 会执行，并输出版本信息示例如下：

```
SDK vX.X XXXX
```

2.2.1.2 命令码 0x01 激活 API

	偏移（字节）	*大小（字节）*	*说明*
请求数据	**0**	4	appid，服务器注册时候生成的内容
	4	4	api_level，API 权限级别
	8	4	app_ver，用户程序版本
	12	32	bundle_id， App 的唯一 ID
应答数据	**0**	2	返回码，应答码 0：成功 1：参数非法，参数长度不匹配 2：数据包加密了，未能正确识别 3：没有激活过的设备，尝试激活 4：DJI App 没有响应，可能是没有连接 DJI App 5：DJI App 没有联网 6：服务器拒绝了，激活失败 7：权限级别不够

推荐发送命令数据的 C/C++结构体

```
typedef struct // 1 byte aligned
{
    unsigned int app_id;
    unsigned int app_api_level;
    unsigned int app_ver;
    unsigned char  app_bundle_id[32];
} activation_data_t;
```

备注：文档中介绍的结构体示例都要求 1 字节对齐。开发者需要根据自身的开发编程环境及编程语言保证结构体的对齐方式为 1 字节对齐。

推荐接收应答数据的 C/C++枚举类型为：

```
enum ErrorCodeForActivatie
{
    errActivateSuccess,
    errActivateInvalidParamLength,
    errActivateDataIsEncrypted,
    errActivateNewDevice,
    errActivateDJIAppNotConnected,
    errActivateDJIAppNoInternet,
    errActivateDJIServerReject,
    errActivateLevelError
};
```

假设激活 API 命令的回调函数为：

```
void activation_callback(const void* p_data, unsigned int n_size)
```

```
{
}
```

应用程序中发送激活 API 命令的操作如下：

```
unsigned char cmd_buf[46];
activation_data_t activation_request_data;
//USER TODO…
//activation_request_data.app_id = 0x0;
//activation_request_data.app_api_level= 0x0;
//activation_request_data.app_ver= 0x0;
//memset(activation_request_data.app_bundle_id,0,32)
cmd_buf[0] = 0x00;//command set
cmd_buf[1] = 0x01;//command id
memcpy(
(void*)&cmd_buf[2],
(void *)&activation_request_data),
sizeof(activation_data_t)
);
Linklayer_Send(SESSION_MODE3,
               cmd_buf,
               46,
               0,
               200,
               3,
               activation_callback
);
```

以上使用的会话方式 3 进行激活 API 请求，飞机收到请求并响应后，应用程序相应的回调函数 activation_callback 会执行，可以判断是否激活成功。

2.2.1.3 命令码 0xFE 透传数据（机载设备至移动设备）

机载设备发送给移动的数据包。最大包大小为 100 字节，带宽约 8KB/s。

	偏移（字节）	*大小（字节）*	*说明*
请求数据	**0**	1~100	用户自定义数据
应答数据	**0**	2	返回码，应答码 0：成功

```
char cmd_buf[10];
cmd_buf[0] = 0x00;
cmd_buf[1] = 0xFE;
memcpy(&cmd_buf[2], “Hello!”, 7);
Linklayer_Send(
                                SESSION_MODE3,
                                cmd_buf,
                                9,
```

```
                                    0,
                                    200,
                                    3,
                                    0
);
```

2.2.2 命令集 0x01 飞行控制类

2.2.2.1 命令码 0x00 请求获得控制权

	偏移（字节）	*大小（字节）*	*说明*
请求数据	**0**	1	1：请求获得控制权 0：请求释放控制权
应答数据	**0**	2	返回码 0x0001：成功释放控制权 0x0002：成功获得控制权 0x0003：获得控制权失败

飞机可以接受三种设备的控制输入：遥控器、移动设备、机载设备而。三种设备的控制输入的优先级最大是遥控器，其次是移动设备，优先级最低是机载设备。
假设请求获得控制权命令的回调函数为：

```
void get_control_callback(const void* p_data, unsigned int n_size)
{
}
```

应用程序中发送激活 API 命令的操作如下：

```
unsigned char cmd_buf[46];
cmd_buf[0] = 0x01;//command set
cmd_buf[1] = 0x00;//command id
cmd_buf[2] = 0x01;//get control
Linklayer_Send(SESSION_MODE3,
               cmd_buf,
               3,
               1,
               200,
               3,
               get_control_callback
);
```

以上使用的会话方式 3 进行获取控制权请求，飞机收到请求并响应后，应用程序相应的回调函数 get_control_callback 会执行，可以判断是否成功。

备注：获得控制权请求需在激活成功后进行，激活成功后，机载设备和飞机的通信必须采用密文通信。

2.2.2.2 命令码 0x01-0x02 状态控制命令

机载设备对飞机的状态控制分为两个阶段。
第一个阶段是发送命令码为 0x01 的状态控制指令。

	偏移（字节）	*大小（字节）*	*说明*
请求数据	**0**	1	指令序列号
	1	1	控制 1：请求进入自动返航 4：请求自动起飞 6：请求自动降落
应答数据	**0**	1	返回码 0x0001：执行失败 0x0002：开始执行

飞机收到状态控制指令之后会立即发送表明已经收到指令的应答数据包，正常情况飞机返回表示“开始执行”应答数据；但如果飞控正在执行一条之前的指令，则返回“执行失败”的应答数据。飞控开始执行指令之后会尝试切换状态模式，并把执行成功与否的结果保存下来。

第二个阶段是机载设备在发送状态控制指令之后可以开始尝试发送命令码为 0x02 的执行结果查询命令。

	偏移（字节）	*大小（字节）*	*说明*
请求数据	**0**	1	指令序列号
应答数据	**0**	1	返回码 0x0001：执行失败（指令序列号不是当前执行的指令） 0x0003：指令正在执行 0x0004：指令执行失败 0x0005：指令执行成功

2.2.2.3 命令码 0x03 姿态控制命令

	偏移（字节）	*大小（字节）*	*说明*
请求数据	**0**	1	模式标志位①
	1	4	roll 方向控制量或 X 轴控制量
	5	4	pitch 方向控制量或 Y 轴控制量
	9	4	yaw 方向控制量（偏航）
	13	4	throttle 方向控制量或 Z 轴控制量
应答数据	**0**		无应答数据

①：详细说明参考飞控控制附加说明章节

推荐发送姿态控制命令数据的 C/C++结构体

```
typedef struct // 1 byte aligned
```

```
{
    unsigned char ctrl_flag;
    float    roll_or_x;
    float pitch_or_y;
    float    yaw;
    float throttle_or_z;
}control_input;
```

备注：文档中介绍的结构体示例都要求 1 字节对齐。开发者需要根据自身的开发编程环境及编程语言保证结构体的对齐方式为 1 字节对齐。

根据模式标志位的值，四个输入控制量会被解释成不同含义的输入，有些情况下是 Body 坐标系，有些情况下是 Ground 坐标系。关于坐标系和模式标志位的说明请参考附述章节。

2.2.3 命令集 0x02 飞控外发的数据

2.2.3.1 命令码 0x00 标准数据包

飞控外发的状态数据包可以通过 DJI N1 PC 调参软件配置。可以配置状态包是否发送及发送的频率。

	偏移（字节）	*大小（字节）*	*说明*
推送数据	**0**	2	状态包存在标志位 bit 0：时间戳包存在标志 bit 1：姿态四元素包存在标志 bit 2：Ground 坐标系下的加速度包存在标志 bit 3：Ground 坐标系下的速度包存在标志 bit 4：Body 坐标系的角速度包存在标志 bit 5：GPS 位置、海拔（气压计数值）、相对地面高度、健康度包存在标志 bit 6：磁感计数值包存在标志 bit 7：遥控器通道值包存在标志 bit 8：云台 roll、pitch、yaw 数据包存在标志 bit 9：飞行状态包存在标志 bit 10：剩余电池百分比包存在标志 bit 11：控制设备包存在标志 *bit [12:15]：保留不用* 标志位为 1 表示标准数据包中存在该状态包
	2	4	时间戳
	6①	16	姿态四元素
	22	12	Ground 坐标系下的加速度
	34	12	Ground 坐标系下的速度

	偏移（字节）	*大小（字节）*	*说明*
	46	12	Body 坐标系的角速度
	58	24	GPS 位置 海拔（气压计数值） 相对地面高度
	82	12	磁感计数值
	94	10	遥控器通道值
	104	12	云台 roll、pitch、yaw 数据
	116	1	飞行状态
	117	1	剩余电池百分比
	118	1	控制设备
应答数据	**0**		无应答数据

①：第一个状态包是时间戳包，之后的状态包偏移字节是不固定的，根据该状态包之前的状态包是否发送而定。

标准数据包中各个状态包的数据段含义如下表所示：

标志数据包				
状态包	状态包字段	数据段类型	说明	默认频率
时间戳	time	unsigned int	时间戳（时间间隔 1/600s）	100Hz
姿态四元素	q0	float	姿态四元数（从 Ground 坐标系转到 Body 坐标系）	100Hz
	q1	float		
	q2	float		
	q3	float		
Ground 坐标系下的加速度	agx	float		100Hz
	agy	float		
	agz	float		
Ground 坐标系下的速度	vgx	float		100Hz
	vgy	float		
	vgz	float		
Body 坐标系下的角速度	wx	float		100Hz
	wy	float		
	wz	float		
GPS 位置、海拔、相对地面高度、信号健康度	longti	double	GPS 位置	100Hz
	lati	double		
	alti	float	海拔（气压计数值）	
	height	float	相对地面高度（超声波和气压计融合）	
	health_flag	unsigned char	GPS 健康度 (0-5, 5 为最好)	
磁感计	mx	float	磁感计数值	0Hz
	my	float		
	mz	float		

遥控器数据	roll	signed short	遥控通道 roll 数据	50Hz
	pitch	signed short	遥控通道 pitch 数据	
	yaw	signed short	遥控通道 yaw 数据	
	throttle	signed short	遥控通道 throttle 数据	
	mode	signed short	遥控通道 mode 数据（模式选择开关）	
	gear	signed short	遥控通道 gear 数据（正面的圆形拨杆）	
云台状态数据	roll	float	云台 roll 数据	50Hz
	pitch	float	云台 pitch 数据	
	yaw	float	云台 yaw 数据	
飞行状态	status	unsigned char	飞行状态	10Hz
电量	status	unsigned char	剩余电量百分比	1Hz
控制设备	status	unsigned char	控制设备 0：遥控器 1：移动设备 2：机载设备	0Hz

机载设备端可以按照如下 C/C++程序示例接收飞控外发的包含飞机状态的标准数据包

```
typedef struct
{
    float  q0;
    float  q1;
    float  q2;
    float  q3;
}sdk_q_data_t;

typedef struct
{
    float  x;
    float  y;
    float  z;
}sdk_common_data_t;

typedef struct
{
    double lati;
    double longti;
    float  alti;
    float  height;
    unsigned char health_flag
}sdk_gps_height_data_t;

typedef struct
{
```

```
    signed short   roll;
    signed short   pitch;
    signed short   yaw;
    signed short   throttle;
    signed short   mode;
    signed short   gear;
}sdk_rc_data_t;

typedef struct
{
    signed short   x;
    signed short   y;
    signed short   z;
}sdk_mag_data_t;

typedef struct // 1 byte aligned
{
    unsigned int      time_stamp;
    sdk_q_data_t   q;
    sdk_common_data_t a;
    sdk_common_data_t v;
    sdk_common_data_t w;
    sdk_gps_height_data_t pos;
    sdk_mag_data_t mag;
    sdk_rc_data_t  rc;
    sdk_common_data_t gimbal;
    unsigned char  status;
    unsigned char  battery_remaining_capacity;
    unsigned char  ctrl_device;
}sdk_std_data_t;

#define _recv_std_data(_flag, _enable, _data, _buf, _datalen) \
    if( (_flag & _enable))\
    {\
    memcpy((unsigned  char*)&(_data),(unsigned  char*)(_buf)+(_datalen),\
    sizeof(_data));\
        _datalen += sizeof(_data);\
    }

static sdk_std_data_t recv_sdk_std_data = {0};
void recv_std_package(unsigned char* pbuf,unsigned int len)
{
    unsigned short *valid_flag = (unsigned short *)pbuf;
    unsigned short data_len = 2;
    _recv_std_data(
    *valid_flag,0x0001,recv_sdk_std_data.time_stamp,pbuf, data_len
```

```
  );
    _recv_std_data(*valid_flag,0x0002 ,recv_sdk_std_data.q,pbuf,data_len
);
    _recv_std_data(*valid_flag,0x0004,recv_sdk_std_data.a
    ,pbuf,data_len);
    _recv_std_data(
  *valid_flag,0x0008_MSG_V,recv_sdk_std_data.v,pbuf,data_len
  );
  _recv_std_data(*valid_flag,0x0010,recv_sdk_std_data.w
    ,pbuf,data_len);
    _recv_std_data(*valid_flag,0x0020,recv_sdk_std_data.pos,pbuf,data_le
n);
    _recv_std_data(*valid_flag,0x0040,recv_sdk_std_data.mag,pbuf,data_le
n);
 _recv_std_data(*valid_flag,0x0080,recv_sdk_std_data.rc,pbuf,data_len);
    _recv_std_data(
  *valid_flag,0x0100,recv_sdk_std_data.gimbal,pbuf, data_len
  );
  _recv_std_data(
  *valid_flag,0x0200,recv_sdk_std_data.status,pbuf, data_len
  );
    _recv_std_data(
  *valid_flag,0x0400,recv_sdk_std_data.battery_remaining_capacity,pbuf,d
ata_len
  );
    _recv_std_data(
  *valid_flag,0x0800,recv_sdk_std_data.ctrl_device ,pbuf, data_len
  );
}
```

备注：文档中介绍的结构体示例都要求 1 字节对齐。开发者需要根据自身的开发编程环境及编程语言保证结构体的对齐方式为 1 字节对齐。

2.2.3.2 命令码 0x01 控制权归属切换

机载设备的控制权优先级最低，其控制权可能在任何时候被夺去。控制权归属切换数据包会在机载控制权被夺去的时由飞控主动推送。

	偏移（字节）	*大小（字节）*	*说明*
推送数据	**0**	1	数据值固定为 0x04
应答数据	**0**	0	无应答数据

2.2.3.3 命令码 0x02 透传数据（移动设备至机载设备）

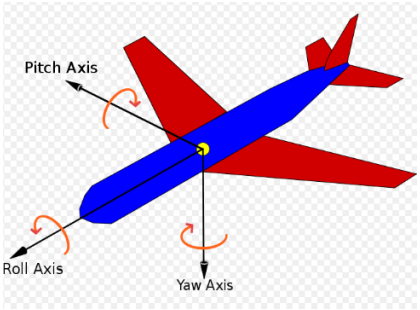
移动设备发送给机载设备的数据包。最大包大小为 100 字节，带宽约 1KB/s。

	偏移（字节）	*大小（字节）*	*说明*
推送数据	**0**	1~100	用户自定义数据
应答数据	**0**	0	无应答数据

3 飞行控制附加说明

3.1 坐标系说明

- Body 坐标系：



- Ground 坐标系（北东地坐标系）：

北-x

东-y

指向地心-z

坐标满足右手定则。ground 坐标系下通用的航向定义是以北为 0，顺时针到 180 度，逆时针到-180 度。这样用-180 到 180 度的数值表示飞行平台在空间中的朝向。

备注：Ground 坐标系的高度方向与人对飞行控制的直觉不符，因此我们将竖直方向的高度和速度都调整成了以天空方向为正，也即发送数值为正的速度会让飞行平台远离地面。但是调整高度方向并不改变 Ground 坐标系的另外两个轴的方向和顺序。

3.2 模式标志位说明

模式标志位代表不同模式的配置。因为多旋翼的结构特点，飞行控制的时候，要把控制信息分解成三部分，竖直、水平和偏航，每个部分都有几种选择。

类别	模式	说明
竖直方向	VERT_POS	垂直方向上控制的是位置，输入的控制量必须为对地面的高度量
	VERT_VEL	垂直方向上控制的是速度

	VERT_THRUST	垂直方向上控制的是油门百分比
水平方向	HORI_ATTI_TILT_ANG	水平方向控制的是 body 坐标系下 pitch 和 roll 两个方向上的倾角（和加速度对应）
	HORI_POS	水平方向控制的是 pitch 和 roll 两个方向上的位置 offset，可以选择这个 offset 是 ground 坐标系下还是 body 坐标系下
	HORI_VEL	水平方向控制的是 pitch 和 roll 两个方向上的速度，可以选择这个速度是 ground 坐标系下还是 body 坐标系下
偏航	YAW_ANG	偏航控制一个 ground 坐标系下的目标角度
	YAW_RATE	偏航控制目标角速度，可以选择这个角速度是 ground 坐标系下还是 body 坐标系下

	控制位	说明
模式标识位 1byte	bit[7：6]	0b00 ： 水平倾角 0b01 ： 水平速度 0b10 ： 水平位置
	bit[5：4]	0b00 ： 垂直速度 0b01 ： 垂直位置 0b10 ： 垂直推力
	bit[3]	0b0 ： 偏航 YAW 角度 0b1 ： 偏航 YAW 角速度
	bit[2：1]	0b0 ： 水平坐标系为 Ground 系 0b1 ： 水平坐标系为 Body 系
	bit[0]	0b0 ： 偏航坐标系为 Ground 系 0b1 ： 偏航坐标系为 Body 系

在某些模式中，水平坐标系和偏航坐标系可以是任意的。比如当水平方向选择 HORI_ATTI_TILT_ANG 模式时，pitch 和 roll 的输入量一定会被解释成 body 坐标系中角度。

经过多种模式的组合，共有 14 种模式 (模式标志指的是 1byte 标志位中的每个 bit 应该如何取值可以实现该模式。数值为 X 的 bit 说明该模式不判断该位置，因此该 bit 可以为任意值)：

模式编号	组合形式	输入数值范围 （throttle/pitch & roll/yaw）	模式标志
1	VERT_VEL HORI_ATTI_TILT_ANG YAW_ANG	-4m/s – 4m/s -30 度-30 度 -180 度-180 度	0b000000XX
2	VERT_VEL HORI_ATTI_TILT_ANG YAW_RATE	-4m/s – 4m/s -30 度-30 度 -100 度/s-100 度/s	0b000010XX
3	VERT_VEL HORI_VEL YAW_ANG	-4m/s – 4m/s -10m/s-10m/s -180 度-180 度	0b010000XX
4	VERT_VEL HORI_VEL	-4m/s – 4m/s -10m/s – 10m/s	0b010010XX

	YAW_RATE	-100 度/s-100 度/s	
5	VERT_VEL HORI_POS YAW_ANG	-4m/s – 4m/s 米为单位的相对位置，数值无限制 -180 度-180 度	0b100000XX
6	VERT_VEL HORI_POS YAW_RATE	-4m/s – 4m/s 米为单位的相对位置，数值无限制 -100 度/s-100 度/s	0b100010XX
7	VERT_POS HORI_ATTI_TILT_ANG YAW_ANG	0m 到最大飞行高度 -30 度-30 度 -180 度-180 度	0b000100XX
8	VERT_POS HORI_ATTI_TILT_ANG YAW_RATE	0m 到最大飞行高度 -30 度-30 度 -100 度/s-100 度/s	0b000110XX
9	VERL_POS HORI_VEL YAW_ANG	0m 到最大飞行高度 -10m/s – 10m/s -180 度-180 度	0b010100XX
10	VERL_POS HORI_VEL YAW_RATE	0m 到最大飞行高度 -10m/s – 10m/s -100 度/s-100 度/s	0b010110XX
11	VERL_POS HORI_POS YAW_ANG	0m 到最大飞行高度 米为单位的相对位置，数值无限制 -180 度-180 度	0b100100XX
12	VERL_POS HORI_POS YAW_RATE	0m 到最大飞行高度 米为单位的相对位置，数值无限制 -100 度/s-100 度/s	0b100110XX
13	VERT_THRUST HORI_ATTI_TILT_ANG YAW_ANG	0 – 100 （危险，请小心使用） -30 度-30 度 -180 度-180 度	0b001000XX
14	VERT_ THRUST HORI_ATTI_TILT_ANG YAW_RATE	0 - 100（危险，请小心使用） -30 度-30 度 -100 度/s-100 度/s	0b001010XX