DJI Onboard API Document

V1.0.1

2015.06

Content

Quick Start 3

Introduction 3

Key Features of Onboard API 3

General System Description 3

Remote Controller, Onboard API & Mobile API 5

Command Authorization Levels 6

[ROS based] Wireless control of DJI MATRICE 100 6

Hardware Checklist 6

Software Checklist 7

Setup Steps 7

Onboard API Programming Guide 12

1 Protocol Description 12

1.1 Protocol Frame Format 12

1.2 Protocol Frame Explanation 12

1.3 Protocol Data Field Explanation 13

1.4 Session 14

1.5 API Example 15

2 Command Set Explanation 16

2.1 Command Set and Authorization Level 16

2.2 Command Sets 16

3 Additional Explanation for Flight Control 29

3.1 Explanation of Coordinate Frames 29

3.2 Explanation of ctrl mode flag 29

DJI offers two powerful APIs for developers to create custom applications: the Mobile Device API, and the UAV Onboard API. The Mobile Device API is part of the DJI Mobile SDK and lets developers monitor and control the UAV from a mobile device running iOS or Android that is connected to the remote controller. The UAV Onboard API allows developers to monitor and control the UAV from any system wired directly to the UAV's autopilot through the available serial (UART) interface.

This document introduces the onboard API. It consists of two parts. A quick start gives introduction to the main components of the onboard API, and a programming guide describes all development related information.

Quick Start

In the quick start, we first introduce the onboard API and explain some terminology. Then a sample code is used to demonstrate key steps to get started.

Introduction

DJI MATRICE 100 is a specially designed quadrotor UAV that has a wide dock to put various equipment onboard. Its detachable battery compartment, expansion bars, expansion bays and extra XT60 power ports give developer great convenience to design a compact UAV application system. DJI Onboard API is for developers who want to directly control MATRICE 100 via a serial interface.

Key Features of Onboard API

- Reliable Communication
  Session-based link protocol to prevent package loss with 32-bit CRC.
- Flexible Control Inputs
  Different control methods including position, velocity and attitude control
- Configurable Monitoring Data
  Flight data can be obtained with configurable number of items and frequency
- Autonomous Application Oriented
  Flight mode control and flight data are designed to aid autonomous control and navigation

General System Description

The core components are MATRICE 100 and a device that is installed on MATRICE 100. This device (the onboard device) connects to the autopilot of MATRICE 100 (N1 Autopilot) by serial cable. The onboard device can be any small sized computing device that is able to perform serial communication and AES encryption.

DJI N1 PC assistant software can configure MATRICE 100 serial port and upgrade firmware of MATRICE 100. It is a tool similar to other DJI PC-based software. This software is simplified to have only a few

functions that cannot be done by DJI new generation assistant software – DJI Pilot, such as firmware upgrade and serial port configuration.

Due to the safety concern, since onboard API enables developer to implement autonomous UAV systems beyond line-of-sight, DJI has to impose more restricted control registration methods for MATRICE 100. Before using MATRICE 100, a developer must register personal information on dev.dji.com (the DJI server), and then activate MATRICE 100. DJI Server generates App ID and an AES key to the developer. Most part of the communication between the onboard device and MATRICE 100 is encrypted by this key, which reaches MATRICE 100 separately during activation process. The activation and encryption will be explained in detail in “Activation Command Set” section.

Diagram of system structure:

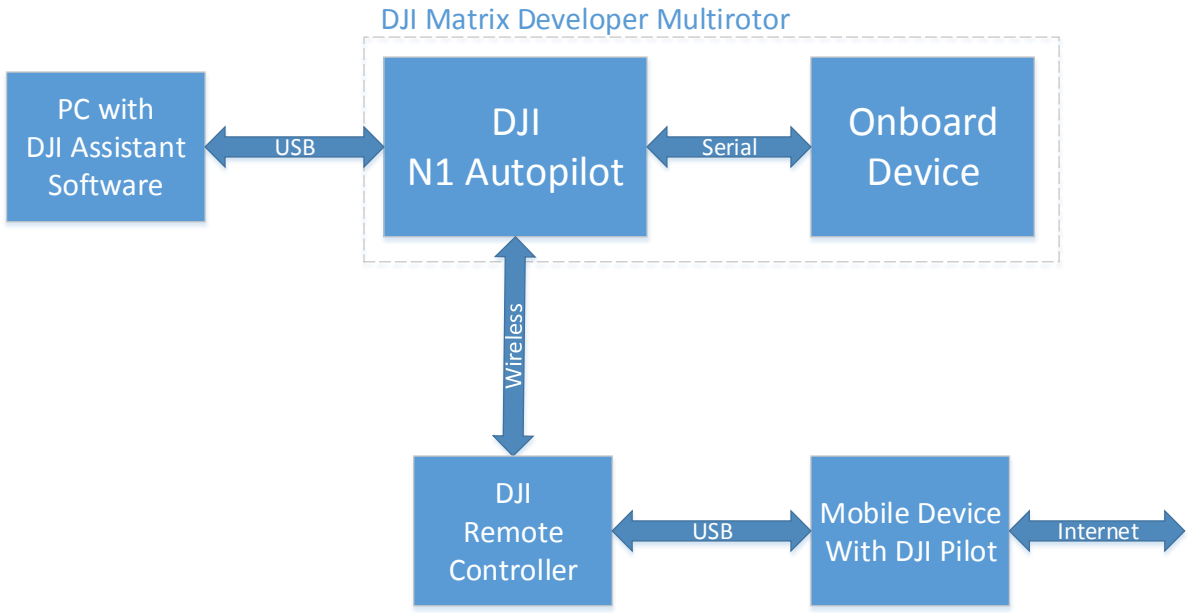
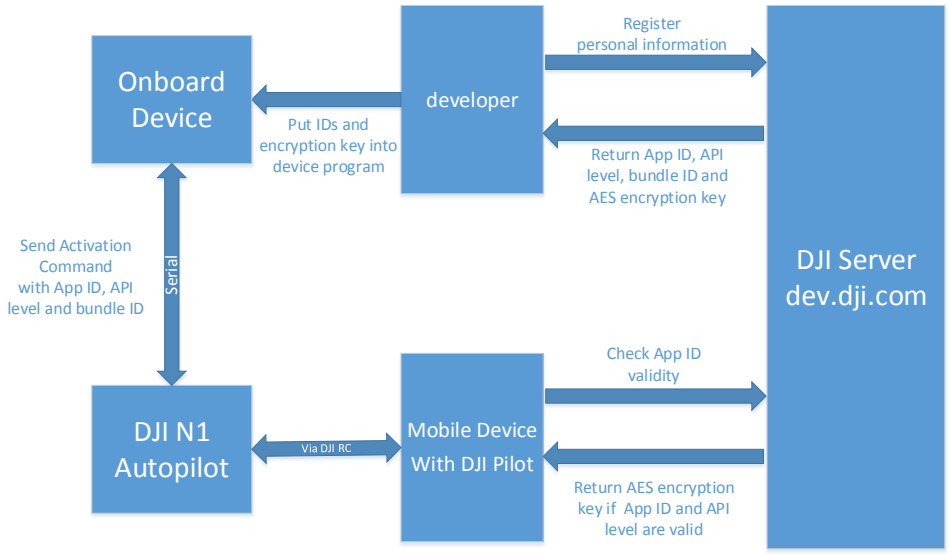


Diagram of registration and activation process:



An important concept in the activation process is the Device Available Number (DAN). It has following properties:

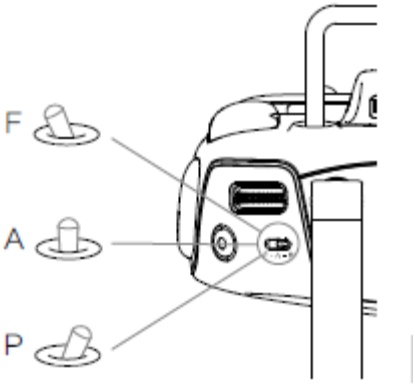
- Each App ID has a DAN. It means the maximum number of autopilots to be activated to support the APP by developer with the same App ID.
- By default, DAN is limited to 5 for a new App ID.
- When autopilot reaches DJI Server during an activation process, the DAN of the App ID increases by 1. If the DAN equals to its limit, new autopilot cannot be activated.
- To increase the limit of the DAN of an App ID, developer shall apply on dev.dji.com.

Remote Controller, Onboard API & Mobile API

DJI Matrice Multirotor is designed to be controlled by Remote Controller (RC), onboard device and mobile device. The standard assistant software "DJI Pilot" for DJI Inspire 1 and Phantom 3 can also be used on the platform. Also DJI Mobile SDK is applicable to the platform, so the platform can be controlled via mobile API (please visit *dev.dji.com* to learn more about DJI Mobile SDK). Since there are three possible control inputs, it is necessary to prioritize them.

RC is designed to be the primary input source. It can decide whether to let the flight control grant control authority to onboard device and mobile device or not.

The F (stands for function) position of the RC can control the flight controller to enter several functions including IOC and API control mode. The flight controller can enter API control mode if following settings are done:



1. “enable API control” box is ticked in PC assistant software (See example below for more details).

2. IOC mode is turned off (Use DJI Pilot App to check this setting).

3. RC mode selection bar to F position (see the above figure).

If RC let the flight controller to enable API control, onboard API and mobile API can request control authority using their API functions. Mobile API is designed to have higher control priority. If mobile API has the control authority, onboard API cannot successfully obtain the control.

This document focuses on introducing the onboard API. It assumes mobile API is not used along with onboard API. In the current version, hybrid control (using both mobile API and onboard API) is not fully supported.

Command Authorization Levels

When a developer register on *dev.dji.com*, an authorization level will be assigned to the developer according to application needs and programming experience of the developer. The developer must save this authorization level (*app_api_level*) to the onboard device. This *app_api_level* will be checked during activation process.

Different authorization levels unlock different commands that the developer can use.

- Level 0, activation related commands
- Level 1, monitor and non-flight control, including camera and gimbal control, flight data monitoring. This level does not involve direct control of craft motion.
- Level 2, flight control. It contains not only motion control, but also some flight mode changing control commands.

In future onboard API, we will release more and more commands with different authorization levels.

[ROS based] Wireless control of DJI MATRICE 100

This example uses our sample code “*dji_keyboard_ctrl*” to control MATRICE 100 remotely. The code is based on ROS package keyboardteleopjs. We have created a simple HTML GUI to help developers to get familiar with controlling MATRICE 100 with keyboard and mouse.

Hardware Checklist

1. DJI Matrice Developer Multirotor MATRICE 100
2. DJI serial cable

(Included in MATRICE 100 accessories)

3. Dupont line 10-20pcs
   (Can be obtained on any electronic component store)
4. 433(434) wireless serial transceiver module 2pcs
   (Reference: http://www.seeedstudio.com/depot/434Mhz-Wireless-Serial-Transceiver-Module-40-Meters-p-1732.html)
5. USB to TTL serial cable 1pc
   (Reference: http://www.adafruit.com/product/954)
   Notice: PL2303 driver is needed to use the USB to TTL serial cable on Windows/Mac.
6. 5V DC-DC Converter
   (Reference: http://www.adafruit.com/products/1385)
   Notice: MATRICE 100 does not provide 5V power, so the serial transceiver module must be powered with external DC-DC converter.

Software Checklist

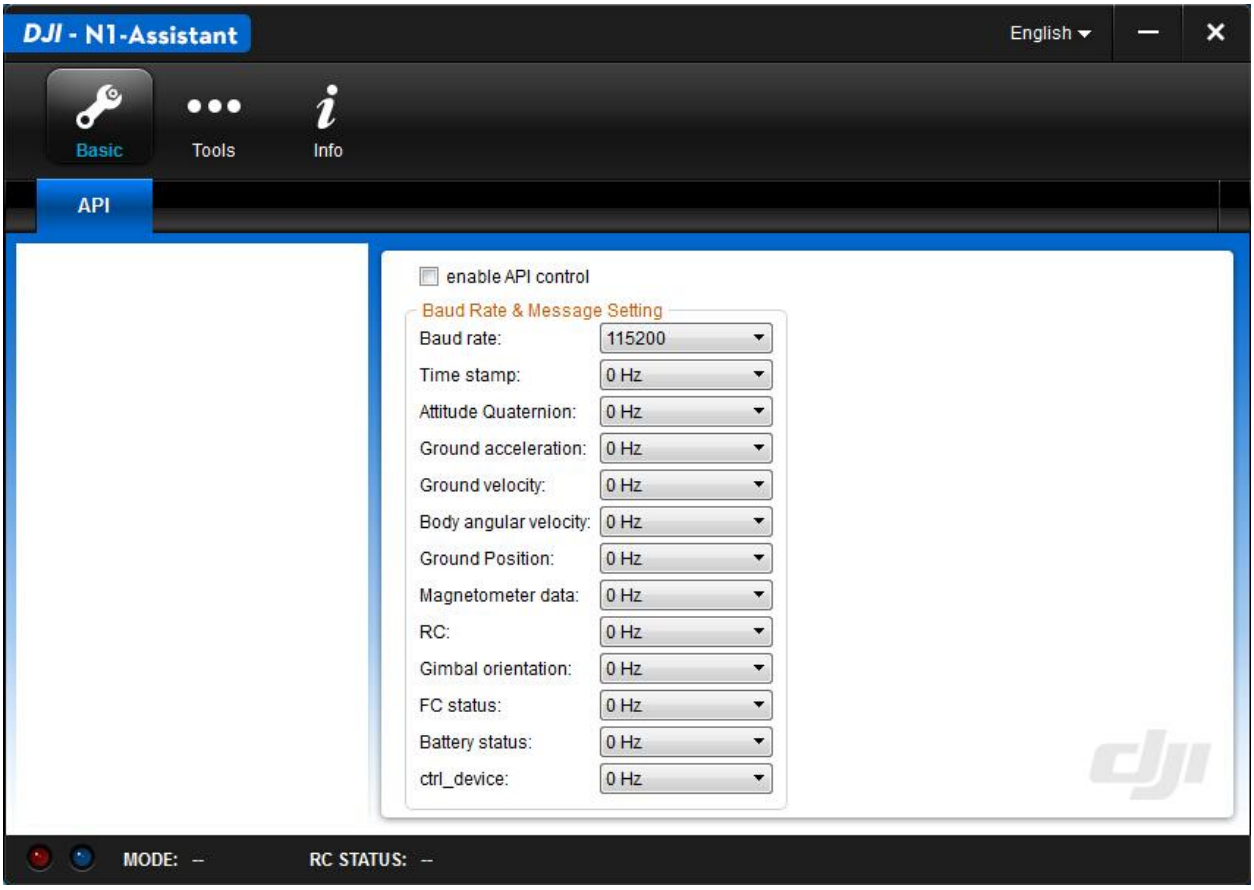
1. Windows PC with DJI N1 PC assistant software
2. Mobile Device with DJI Pilot (newest version) installed. The device must be able to access Internet
3. Linux PC (or embedded device) with Ubuntu 14.04 (or higher) and ROS Indigo (or higher). The sample code is tested on ROS Indigo only.
4. ROS package *rosbridge_server*
5. Sample code "*dji_sdk*" and "*dji_keyboard_ctrl*"

Setup Steps

- Prepare MATRICE 100

  Power on MATRICE 100 and connect it to PC. DJI N1 PC assistant software can let users to update firmware and configure MATRICE 100 to enable API control mode.

  On tab "Basic", developers can tick the "enable API control" box to enable remote controller and MATRICE 100 to have API control related functions. Developers may alter the serial baud rate and data package content in "Baud Rate & Message Setting" area.



After API control mode is enabled, switch the mode selection bar of the remote controller to the middle position (*F position*)

- Establish communication link

  Configure the 433 transceivers to a baud rate of 230400 (different transceivers might require different initialization steps). Connect one transceiver to PC through the USB to TTL cable. Use the DJI serial cable to connect the other transceiver to the autopilot of MATRICE 100. Be careful with the transceiver on MATRICE 100, its power must come from the 5V DC-DC converter, which can draw power from MATRICE 100 battery.

- Run sample code

  1. Compile ROS package dji_sdk
  2. Start *roscore*, and then start *rosbridge_server* in a new terminal page

     ```
     roslaunch rosbridge_server rosbridge_websocket.launch
     ```

  3. Use the launch file in the sample code to start dji_sdk_node.

     Following code is the sample launch file

```
<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>
```

The node parameters are

Name	Type	Explanation
serial_name	String	Serial device name. It usually is "/dev/ttyUSB0", but it may be different depends on Linux machine. "ls /dev/*" and "dmesg | tail" commands can be used to identify device name.
baud_rate	Int	The serial port baud rate. It must be the same as MATRICE 100 configuration.
app_id	Int	The app ID assigned by *dev.dji.com* server to developer while registration.
app_api_level	Int	The app API control level assigned by *dev.dji.com* server to developer while registration.
app_version	Int	Developer assigned application version
app_bundle_id	String	The app bundle ID assigned by *dev.dji.com* server to developer while registration.
enc_key	string	The encryption key assigned by *dev.dji.com* server to developer while registration.

Notice: This command must use root account to start. Because root permission is needed to read serial device.

```
sudo su

roslaunch dji_sdk sdk_demo.launch
```

4. Edit "sdk_keyboard_demo.html", changing the address in the url to Linux machine hostname, localhost or 127.0.0.1.

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

5. Open "sdk_keyboard_demo.html" in web browser. *rosbridge_server* will print log showing that new client is connected. If not, please check the connection settings in step 4. After the

html page is connected to *rosbridge_server*, the web GUI will show some flight status.

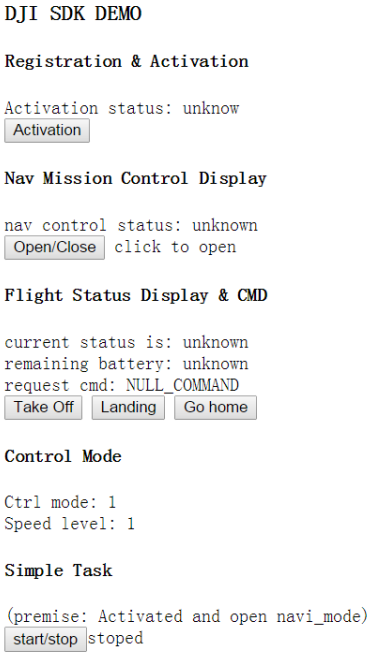
It is also possible to directly check flight status by using *rostopic*.

- Test communication link
  On the web GUI, click button “Activation”. If the communication link is ready to use, MATRICE 100 will acknowledge the GUI. If not, please debug the transceivers or MATRICE 100 settings.

- Activate MATRICE 100 to use API
  Connect the mobile device with the DJI Pilot App to the remote controller of MATRICE 100, and make sure the mobile device has access to the Internet. The activation process is done automatically after clicking the “Activation” button.

- Control MATRICE 100
  The web GUI has control buttons as shown below. Moreover, keyboard “W, A, S, D” can let MATRICE 100 move horizontally, “Z, C” controls vertical velocity, and “Q, E” controls yaw. Developers can try all the functionalities via the web GUI. Please make sure you have enough testing space.

  The horizontal movement is controlled by angle command associated with button “W, A, S, D”. The angle is 5*speed_level. speed_level is an inner variable with default value 1. Its value can be changed by keyboard button 123456. Press different buttons can change the value of angle command. Please be careful when you are using large angle commands. MATRICE 100 will be accelerate rapidly.



- Safety during flight

MATRICE 100 only responses to serial control command when the remote controller's mode selection bar is at its middle position (*F position*). Anytime when user switches the mode selection bar, the API control mode is turned off. We recommend two developers work together during the testing. One developer control the web GUI, while the other developer hold the remote controller for emergency use.

If the user wants to switch back to *F position* and reenter the API mode, onboard application does not need to send control request again to get control authority.

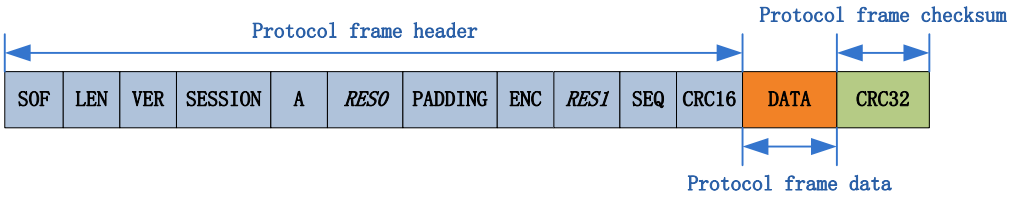
If the mode selection bar is already at *F position* when MATRICE 100 is powered on, then user must switch back and forth to enable API control mode. This mechanism can prevent MATRICE 100 from executing automatic applications without user permission.

Onboard API Programming Guide

This part discusses the necessary knowledge about how to program to communicate with MATRICE 100. We recommend developers follow quick start first to execute our sample code first before reading this programming guide.

1 Protocol Description

1.1 Protocol Frame Format



1.2 Protocol Frame Explanation

Field	Byte Index	Size（bit）	Description
SOF	0	8	Frame start number, fixed to be 0xAA
LEN	1	10	Frame length, maximum length is 1023 bytes
VER		6	Version of the protocol
SESSION	3	5	The session ID used during communication
A		1	Frame Type 0: data 1: acknowledgement
RES0		2	Reserved bits, fixed to be 0
PADDING	4	5	The length of addition data added in link layer. It comes from the encryption process
ENC		3	Frame data encryption type 0：no encryption 1：AES encryption
RES1	5	24	Reserved bits, fixed to be 0
SEQ	8	16	Frame sequence number
CRC16	10	16	Frame header CRC16 checksum
DATA	12	--①	Frame data, maximum length 1007bytes
CRC32	--②	32	Frame CRC32checksum

①： Frame data size can vary, 1007 is the maximum length.

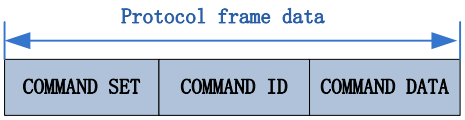
②：The index of this field depends on the length of the data field.

1.3 Protocol Data Field Explanation

All serial packages exchanged between MATRICE 100 and the onboard device can be classified into three types:

1. Command package. From the onboard device to MATRICE 100. It mainly contains flight control commands.
2. Message package. From MATRICE 100 to the onboard device. It contains the autopilot data.
3. Acknowledgement package (Ack package). From MATRICE 100 to the onboard device. It contains execution results of commands.

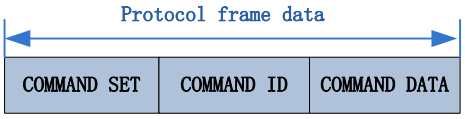
1.3.1 Package from the onboard device to MATRICE 100 (Command Package)



Data Field	Byte Index	Size（byte）	Description
COMMAND SET	0	1	Please look at “2 Command Set Explanation”, control comands
COMMAND ID	1	1	
COMMAND DATA	2	--①	

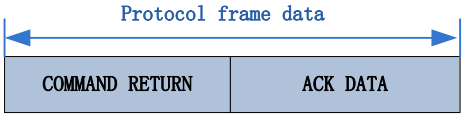
①: Size depends on the exact command used.

1.3.2 Package from the autopilot to onboard device (Message package)



字段	字节索引	大小（单位 byte）	说明
COMMAND SET	0	1	Please look at “2 Command Set Explanation”, monitor command
COMMAND ID	1	1	
COMMAND DATA	2	--	

1.3.3 Package from the autopilot to onboard device (Ack package)



字段	字节索引	大小（单位 byte）	说明
COMMAND RETURN	0	2	The return code of the command result
ACK DATA	2	--	Return data

1.4 Session

An important requirement of autonomous control is the communication reliability. We design a “session mechanism” to make sure command packages and acknowledgement packages are delivered without package loss.

When developer compiles a message package in the onboard device program, a session ID should be used depending on the reliability requirement. Different session IDs correspond to different communication channels, i.e. sessions. Onboard API serial link layer has three types of sessions (for brevity, in the follow table, we use Sender to refer the onboard device, and Receiver be the autopilot of MATRICE 100).

Session Mode	SESSION	Description
Mode 1	0	Sender do not need acknowledgement
Mode 2	1	Sender need acknowledgement, but can tolerate Ack package loss.
Mode 3	2-31	Sender wants to make sure the acknowledgement is reliably sent. For these sessions, Receiver saves the sequence

		number in the command package, and send an Ack package upon receiving it. If Ack package loss happened, Sender may request Receiver again using the same command package with the same sequence number, and Receiver will reply by sending saved acknowledge result. Unless Sender sends a new sequence number, Receiver will not forget last command acknowledge result.

1.5 API Example

Let session mode be represented by the following enum:

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

And define a callback function for handling the return data of the command as:

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

Finally we define the communication function interface:

```
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)
```

Arguments explained:

p_data	The start pointer to the datastream
n_size	The size of datastream
enc_type	Whether this package be encrypted or not
ack_timeout	When using session 3, this parameter decides how long to resend command
retry_time	When using session 3, this parameter decides how many times to rety
cmd_callback	The function pointer to the callback function

Notice: Here a dummy link layer send interface is defined for demonstration purpose. Since Session Mode 3 is reliable, the communication function interface should contain parameters such as length of timeout and number of resending times.

2 Command Set Explanation

2.1 Command Set and Authorization Level

The DJI onboard API has three sets or categories of commands:

Category	Description	Command Set ID
Activation related	All commands used to activate API	0x00
Control related	Commands to control MATRICE 100	0x01
Monitoring related	Commands that contains autopilot data	0x02

Each command set has a unique set ID. All commands belong to one command set have different command ID.

All control commands have an associated authorization level. In the current version, we have opened 5 stable control commands and several unstable commands, which are all level 2. In the standard version and future versions, more control commands will be opened up at different authorization levels. A tentative level schedule is as follows:

API Levels	Brief Plan
0	API Activation commands
1	Camera and gimbal control commands (coming soon)
2	Flight control commands

2.2 Command Sets

2.2.1 Activation Command Set: 0x00

To activate the API, session ID 2-31 can be used to guarantee ack packages are returned. All commands in this command set has authorization level 0, so that all users can use these commands to activate MATRICE 100 or debug connection status. The activation process let MATRICE 100 to connect to Internet via DJI Pilot. A mobile device with Internet connection is needed.

2.2.1.1 Command ID 0x00 get API version

	Offset	*Size*	*Description*
Request data	**1**	1	Arbitrary number
Return Data	**0**	2	return code
	2	4	The CRC code of version string
	6	32	SDK version string

Recommended receiving C/C++ struct

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

Let the callback function of get API version be:

```
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);
    }
}
```

To send get API version package, we can use follow code piece:

```
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
);
```

Session Mode 3 is used to get API version. After the autopilot receives request and responses, function print_sdk_version will be executed and following version information printed:

```
SDK vX.X XXXX
```

2.2.1.2 Command ID 0x01 Activate API

	Offset	*Size*	*Description*
Request data	**0**	4	appid, a number obtained when user register as developer
	4	4	api_level，authorization level
	8	4	app_ver，the version of onboard application
	12	32	bundle_id， the name of the application, set by developer
Return data	**0**	2	Return code 0 Success 1 Invalid parameters 2 Cannot recognize encrypted package 3 Attempt to activate 4 DJI Pilot App no response 5 DJI Pilot App no Internet 6 Server rejected activation attempt 7 Insufficient authority level

Recommended sending C/C++ struct

```
typedef __attribute__((__packed__)) 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;
```

Notice: All the struct in the document requires 1 byte alignment (for example, using typedef __attribute__((__packed__)) struct). Developers must make sure their structs are 1-byte aligned.

Recommended receive C/C++ enum data

```
enum ErrorCodeForActivatie
{
      errActivateSuccess,
```

```
      errActivateInvalidParamLength,
      errActivateDataIsEncrypted,
      errActivateNewDevice,
      errActivateDJIAppNotConnected,
      errActivateDJIAppNoInternet,
      errActivateDJIServerReject,
      errActivateLevelError
};
```

Let the API activation callback function be:

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

To send API activation package, we can use follow code piece:

```
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
);
```

Session Mode 3 is used to activate API. After the autopilot receives request and responses, function activation_callback will be executed, in which developer can check whether API activation is successful or not.

2.2.1.3 Command ID 0XFE Data transparent transmission (From airborne equipment to Mobile device)

The downstream bandwidth from airborne equipment to mobile device is around 8KB/s

	Offset	*Size*	*Description*
Request data	**0**	1~100	User defined data
Return data	**0**	2	Return code 0 Success

```
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 Control Command: Set 0x01

2.2.2.1 Command ID 0x00 Control authority Request

	Offset	*Size*	*Description*
Request Data	**0**	1	1 = request to get control authority 0 = request to release control authority
Return Data	**0**	2	Return Code 0x0001 = successfully released control authority 0x0002 = successfully obtained control authority 0x0003 = control authority failed to change

There are three types of control devices:

1. remote controller
2. mobile device

3. Onboard device.

The control priority is: remote controller > mobile application > onboard device. Mobile application connects to MATRICE 100 via mobile API. A similar control authority request command exists in the mobile API command set. Therefore it is possible that when onboard device requests control authority through serial API, control authority is already been granted to the mobile application, so according to the priority list, onboard device will fail to obtain control. Besides, mobile API control request can interrupt ongoing onboard API control.

In the current version, hybrid control (using both mobile API and onboard API) is not fully supported. Developer should take care of the priority issue when they are developing hybrid control application. A monitoring data CTRL_DEVICE can be used to check the control authority (see Monitor Command Set 0x02).

0x0003 happens when the mode selection bar of the remote controller in not properly configured (see quick start – Remote Controller), or control authority is already obtained by mobile application.

Let the callback function for obtain control authority is:

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

To send control request package, we can use follow code piece

```
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
);
Session Mode 3 is used to obtain control. After the autopilot receives request and responses, function get_control_callback will be executed, in which developer can check whether control authority is successfully changed or not.
```

Notice: Obtain control authority must be done after API activation, and it should be encrypted.

2.2.2.2 Command ID 0x01-0x02 Flight Mode Control

To control the flight mode of MATRICE 100, onboard device should use two commands to make sure the mode changing control works properly.

Firstly, the command 0x01 should be sent to trigger the mode changing logic:

	Offset	*Size*	*Description*
Request Data	**0**	1	Command sequence number
	1	1	Request mode 1 = request go home 4 = request auto take off 6 = request auto landing
Return Data	**0**	1	Return code 0x0001 the command is received and rejected 0x0002 start to execute the command

When MATRICE 100 received command 0x01, immediate ack package contains either 0x0001 “reject” or 0x0002 “start to execute”. If the autopilot is already executing a flight mode command, a “reject” package is sent. In normal case, after sending the “start to execute” package, the autopilot will try to change the flight mode, and make sure the changing is firmly done. The execution result is saved.

The second command is a query from the onboard device to obtain the execution result.

	Offset	*Size*	*Description*
Request Data	**0**	1	Command sequence number
Return Data	**0**	1	Return code 0x0001 query fail, current command is not the query command 0x0003 command is executing 0x0004 command is failed 0x0005 command is success

There two commands, together with the “session mechanism”, can guarantee that a flight control command is executed and its execution result reaches the onboard device reliably.

2.2.2.3 Command ID 0x03 Movement Control

Please read the instructions carefully before sending commands.

	Offset	*Size*	*Description*
Request Data	0	1	Ctrl mode flag（detailed description below）
	1	4	Roll control value or X-axis control value
	5	4	pitch control value or Y-axis control value
	9	4	yaw control value
	13	4	vertical control value or Z-axis control value
Return Data	0		No ack

①: Detailed explanation in “Additional Explanation for Flight Control”

Recommend sending structure in C/C++

```
typedef __attribute__((__packed__)) struct // 1 byte aligned
{
      unsigned char ctrl_flag;
      float  roll_or_x;
      float pitch_or_y;
      float  yaw;
      float throttle_or_z;
}control_input;
```

Notice: All the struct in the document requires 1 byte alignment (for example, using typedef __attribute__((__packed__)) struct). Developers must make sure their structs are 1-byte aligned.

Notice that depending on the value of ctrl_flag, the four control inputs can have different meanings and represent control inputs in either body frame or ground frame. In the “Additional Explanation for Flight Control”, body frame, ground frame and ctrl_flag are elaborated.

2.2.3 Monitor Command Set: 0x02

2.2.3.1 Command ID 0x00 Message Package

Message package content and frequency can be configured by DJI N1 assistant software. Each data item in the message package has individual frequency. The maximum message frequency is equal to the message’s highest data item update frequency.

	Offset（byte）	***Size（byte）***	***Explanation***
Send Data	**0**	2	Item presence flag bit 0: flag of time stamp bit 1: flag of attitude quaternion bit 2: flag of linear acceleration in ground frame bit 3: flag of linear velocity in ground frame bit 4: flag of angular velocity in body frame bit 5: flag of GPS location, altitude and healthness bit 6: flag of magnetometer bit 7: flag of remote controller data bit 8: flag of gimbal yaw, pitch, roll bit 9: flag of flight status bit 10: flag of battery info bit 11: flag of ctrl device *bit [12:15]: reserved* Bit with value 1 means the message package contains corresponding data item

	Offset（byte）	*Size（byte）*	*Explanation*
	2	4	Time stamp
	6①	16	Attitude quaternion item
	22	12	Linear acceleration item
	34	12	Linear velocity item
	46	12	Angular velocity item
	58	24	GPS position, altitude, height and healthiness
	82	12	Magnetometer data
	94	10	Remote controller channels
	104	12	Gimbal yaw, pitch, roll
	116	1	Flight status
	117	1	Battery percentage
	118	1	Ctrl device
Return Data	**0**		No return data

①：The first data item is time stamp. Following data items may or may not present in the message packages, so their offsets are not fixed. Here we only list the offsets when all data items are sent.

Each data item in the message package is explained below:

Data Item list				
Item Name	Variables	Type	Description	Default frequency
Time	time_stamp	uint32_t	Time in tick（tick interval 1/600s）	100Hz
Q	q0	float32	Attitude quaternion （From ground to body frame）	100Hz
	q1	float32		
	q2	float32		
	q3	float32		
ACC	agx	float32	Linear acceleration in ground frame	100Hz
	agy	float32		
	agz	float32		
VEL	vgx	float32	Linear velocity in ground frame	100Hz
	vgy	float32		
	vgz	float32		
W	wx	float32	Angular velocity in body frame	100Hz
	wy	float32		

	wz	float32		
POS	longti	double	GPS location	100Hz
	lati	double		
	alti	float32	Altitude (measured by barometer)	
	height	float32	Height to ground （barometer, may fuse with ultrasonic if sensor added）	
	health_flag	uint8_t	GPS healthiness (0-5, 5 is the best conditon)	
MAG	mx	float32	Magnetometer readings	0Hz
	my	float32		
	mz	float32		
RC	roll	int16_t	Remote controller roll channel	50Hz
	pitch	int16_t	Remote controller pitch channel	
	yaw	int16_t	Remote controller yaw channel	
	throttle	int16_t	Remote controller throttle channel	
	mode	int16_t	Remote controller mode channel	
	gear	int16_t	Remote controller gear channel	
GIMBAL	roll	float32	Gimbal roll data	50Hz
	pitch	float32	Gimbal pitch data	
	yaw	float32	Gimbal yaw data	
FLIGHT_STATUS	status	uint8_t	Flight status	10Hz
BATTERY	status	uint8_t	Battery percentage	1Hz
CTRL_DEVICE	status	uint8_t	Current control device 0 -> RC 1 -> APP 2 -> onboard device	0Hz

Onboard device can use following code piece to receive the standard message package sent by the autopilot

```
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;
    short   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_len);
      _recv_std_data(*valid_flag,0x0040,recv_sdk_std_data.mag,pbuf,data_len);
 _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,data_len
  );
      _recv_std_data(
  *valid_flag,0x0800,recv_sdk_std_data.ctrl_device,pbuf, data_len
  );
}
```

Notice: All the struct in the document requires 1 byte alignment (for example, using `typedef __attribute__((__packed__)) struct`). Developers must make sure their structs are 1-byte aligned.

2.2.3.2 Command ID 0x01 Control Authority Change Notification

Onboard device has the lowest control priority. Its control authority can be taken over by RC or mobile device at any time. This notification message will be sent to the onboard device by the autopilot at the time when control authority changed.

	Offset（byte）	*Size（byte）*	*Explanation*
Send Data	**0**	1	Fixed number 0x04
Return Data	**0**	0	No return data

2.2.3.3 Command ID 0X02 Data transparent transmission (From Mobile device to airborne equipment)

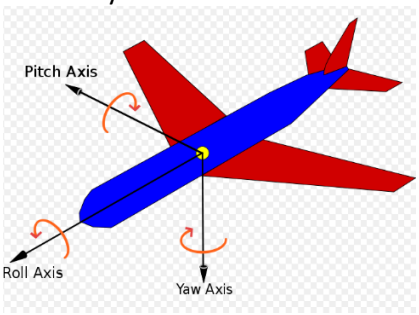
The upstream bandwidth from mobile device to airborne equipment is around 1KB/s

	Offset	*Size*	*Description*
Request data	**0**	1~100	User defined data
Return data	**0**	0	No return data

3 Additional Explanation for Flight Control

3.1 Explanation of Coordinate Frames

- Body frame



- Ground frame

  North – x axis

  East – y axis

  Down – z axis

So in the ground frame, a general definition for craft orientation is North = 0 degree, East = 90 degree, West = -90 degree, and south be either 180 degree or -180 degree.

Notice: The direction of the ground frame is not natural for height control. So we adjust the direction of vertical control to let height and vertical velocity to be positive upwards, in other words, positive velocity makes MATRICE 100 to ascend. This adjustment does not change the direction and the order of the other two axes.

3.2 Explanation of ctrl mode flag

To control the spatial movement of MATRICE 100, we split control inputs into three parts: horizontal control, vertical control and yaw control. Each part has several sub modes.

Category	Mode	Explanation
Vertical	VERT_POS	Control the height of MATRICE 100
	VERT_VEL	Control the vertical speed of MATRICE 100, upward is positive
	VERT_THRUST	Directly control the thrust of MATRICE 100
Horizontal	HORI_ATTI_TILT_ANG	Pitch & roll angle, referenced to the body frame
	HORI_POS	Offsets of pitch & roll directions， referenced to either the ground or body frame

	HORI_VEL	Input are velocities on pitch & roll directions， offset can be chosen in either the ground frame or body frame
Yaw	YAW_ANG	Yaw angle referenced to the ground frame
	YAW_RATE	Yaw angular rate. It can either be referenced to either the ground or body frame.

The ctrl mode flag is divided into 8 bits

	Bit position	Explanation
ctrl_mode_flag 1byte	bit[7：6]	0b00： horizontal angle 0b01： horizontal velocity 0b10： horizontal position
	bit[5：4]	0b00： vertical velocity 0b01： vertical position 0b10： vertical thrust
	bit[3]	0b0： yaw angle 0b1： yaw angular rate
	bit[2：1]	0b0： Horizontal frame is ground frame 0b1： Horizontal frame is body frame
	bit[0]	0b0： Yaw frame is ground frame 0b1： Yaw frame is body frame

HORI_FRAME and YAW_FRAME can be an arbitrary value if the corresponding mode does not need to specify frame. For instance, HORI_ATTI_TILT_ANG defines pitch and roll angles in body frame regardless of input.

By specifying ctrl_mode_flag, 14 control modes can be constructed (ctrl_mode_flag is represented as an 8-bit binary number here. The bit position with X means that this certain mode doesn't depend on that bit position, so its value can be either 0 or 1):

No.	Combinations	Input data range (throttle/ pitch&roll/ yaw)	ctrl_mode_flag
1	VERT_VEL HORI_ATTI_TILT_ANG YAW_ANG	-4m/s – 4m/s -30 degree-30 degree -180 degree -180 degree	000000XX
2	VERT_VEL HORI_ATTI_TILT_ANG YAW_RATE	-4m/s – 4m/s -30 degree -30 degree -100 degree /s-100 degree /s	000010XX
3	VERT_VEL HORI_VEL YAW_ANG	-4m/s – 4m/s -10m/s-10m/s -180 degree -180 degree	010000XX
4	VERT_VEL HORI_VEL	-4m/s – 4m/s -10m/s-10m/s	010010XX

	YAW_RATE	-100 degree /s-100 degree /s	
5	VERT_VEL HORI_POS YAW_ANG	-4m/s – 4m/s Offset in meters (no limit) -180 degree -180 degree	100000XX
6	VERT_VEL HORI_POS YAW_RATE	-4m/s – 4m/s Offset in meters (no limit) -100 degree /s-100 degree /s	100010XX
7	VERT_POS HORI_ATTI_TILT_ANG YAW_ANG	0m to height limit -30 degree -30 degree -180 degree -180 degree	000100XX
8	VERT_POS HORI_ATTI_TILT_ANG YAW_RATE	0m to height limit -30 degree -30 degree -100 degree /s-100 degree /s	000110XX
9	VERL_POS HORI_VEL YAW_ANG	0m to height limit -10m/s – 10m/s -180 degree -180 degree	010100XX
10	VERL_POS HORI_VEL YAW_RATE	0m to height limit -10m/s – 10m/s -100 degree /s-100 degree /s	010110XX
11	VERL_POS HORI_POS YAW_ANG	0m to height limit Offset in meters (no limit) -180 degree -180 degree	100100XX
12	VERL_POS HORI_POS YAW_RATE	0m to height limit Offset in meters (no limit) -100 degree /s-100 degree /s	100110XX
13	VERT_THRUST HORI_ATTI_TILT_ANG YAW_ANG	0 – 100 (use with precaution) -30 degree -30 degree -180 degree -180 degree	001000XX
14	VERT_ THRUST HORI_ATTI_TILT_ANG YAW_RATE	0 – 100 (use with precaution) -30 degree -30 degree -100 degree /s-100 degree /s	001010XX