

# GDU PSDK 功能说明文档

## 什么是 PSDK

为方便您使用 PSDK 提供的新功能开发出安全可靠的负载设备，请持续关注 GDU PSDK 的版本发布信息，及时使用最新版本的 PSDK 开发包开发负载设备。

### 本文所指

- “移动端 App” 为使用 MSDK 开发的移动端 App 或 GduFlight2。
- “负载设备” 为使用 PSDK 开发的负载设备。

## PSDK 介绍

GDU 为支持开发者开发出可挂载在 GDU 无人机上的负载设备，提供了开发工具包 Payload SDK（即 PSDK）以及开发配件，方便开发者利用 GDU 无人机上如电源、通讯链路及状态信息等**资源**。开发者能够根据行业的应用需求，基于 PSDK 提供的功能接口，结合具体的结构设计、硬件设计、软件逻辑实现和算法优化，开发出如**自动巡检系统、红外相机、测绘相机、多光谱相机、喊话器、探照灯**等满足不同细分领域的负载设备。



## 主要优势

- 功能丰富且完善

通过使用 PSDK 提供的如信息获取、数据传输和电源管理等基础功能，以及相机、云台、负载协同和精准定位等高级功能，开发者能够根据行业的应用需求，设计出**功能完善**的负载设备。并且 PSDK 提供了丰富的接口，方便开发者使用第三方应用程序和算法框架，使用图像识别、自主巡航及 SLAM 等技术开发出专业的应用软件，此外，还方便开发者接入第三方传感器、相机或检测设备，采集所需的数据信息，满足用户个性化的应用功能和控制需求。

## PSDK 功能概览

## 日志管理

拥有日志管理功能的负载设备支持用户通过串口、终端或 USB 等方法输出不同模块的

日志信息，使用具有日志颜色显示功能的工具能够以不同的颜色显示不同类型的日志。

## 信息管理

- 无人机系统信息获取：负载设备通过该功能可主动获取 GDU 无人机的型号、硬件平台的类型、移动端 App 使用的语言等信息。
- 消息订阅：负载设备能够订阅无人机上各个部件实时产生的传感器数据以及无人机系统状态信息，如姿态四元数、融合海拔高度及 RTK 位置等。

## 相机功能

- 基础功能：设置相机模式、拍照、录像、获取相机状态

## 云台功能

- 控制云台转动速度

## 时间同步

- 获取 PPS 数据：获取无人机的硬件触发脉冲信号
- 获取 UTC 时间：获取统一的 UTC 时间

## 选择开发平台

请根据操作系统和开发平台对 PSDK 功能的支持差异、负载设备程序的资源占用情况以及 PSDK 支持的工具链，选择开发负载设备的操作系统和开发平台。

## 选择操作系统

## 平台功能差异

功能名称	平台支持-Linux	平台支持-RTOS
日志管理	✓	✓
信息管理	✓	✓
基础相机功能	✓	✓
云台功能	✓	✓
自定义控件	✓	✓
时间同步	✓	✓
精准定位	✓	✓

## 资源占用

### Linux

程序运行时的资源占用情况如下所示：

- 栈：约 12288 字节

- 堆：约 40960 字节
- Text 段：755359 字节
- Data 段：3872 字节
- Bss 段：23848 字节
- CPU 占用：7.2 %

## RTOS

使用 STM32F407VGT6 运行 RTOS 平台 PSDK 示例程序，程序运行时的资源占用情况如下所示：

- Text 段：353620 字节
- Data 段：1836 字节
- Bss 段：85620 字节
- CPU 占用：30 %

## 选择开发平台

PSDK 支持使用如下工具编译基于 PSDK 开发的负载设备，请根据选用的**开发平台**正确地选择工具链。

**说明：** 有关跨平台移植的详细说明请参见

工具链名称	目标平台	典型芯片型号	推荐开发平台
aarch64-linux-gnu-gcc	aarch64-linux-gnu	NVIDIA Jetson TX2、Rockchip RK3399 pro	Manifold2-G、瑞芯微 Toybrick 开发板

x86_64-linux-gnu-gcc	x86_64-linux-gnu	64 位 intel 处理器，如 Intel Core i7-8550U	Maniflod2-C
arm-linux-gnueabi-gcc	arm-linux-gnueabi	ZYNQ、I.MX6Q	-
arm-linux-gnueabihf-gcc	arm-linux-gnueabihf	支持硬件浮点运算的处理器，如 OK5718-C 等	-
armcc-cortex-m4	Cortex M4/M4F 系列 MCU	STM32F407IGT6、STM32F405RGT6	STM32F407-Eval、STM32F407 探索者开发板等
arm-none-eabi-gcc	Cortex M4/M4F 系列 MCU	STM32F407IGT6、STM32F405RGT6	STM32F407-Eval、STM32F407 探索者开发板等
arm-linux-androideabi-gcc	arm-linux-androideabi	高通骁龙系列芯片	安卓平台
aarch64-linux-android-gcc	aarch64-linux-android	高通骁龙系列芯片	安卓平台
arm-himix100-linux-gcc	arm-himix100-linux	hi3516EV 系列芯片	-
arm-himix200-linux-gcc	arm-himix200-linux	hi3516C 系列芯片	-
aarch64-himix100-linux-gcc	aarch64-himix100-linux	hi3559C	-
arm-hisiv300-linux-uclibcgnueabi-gcc	arm-hisiv300-linux-uclibcgnueabi	hi3516A 系列芯片	-
arm-hisiv400-linux-gnueabi-gcc	arm-hisiv400-linux-gnueabi	hi3516A 系列芯片	-
arm-hisiv500-linux-uclibcgnueabi-gcc	arm-hisiv500-linux-uclibcgnueabi	hi3519 系列芯片	-
arm-hisiv600-linux-gnueabi-gcc	arm-hisiv600-linux-gnueabi	hi3519 系列芯片	-

xtensa-esp32-elf-gcc	xtensa-esp32-elf	ESP32 系列芯片	ESP32-DevkitC
----------------------	------------------	------------	---------------

**说明：** 开发者需根据所使用的开发平台，选择指定编译链的静态库。

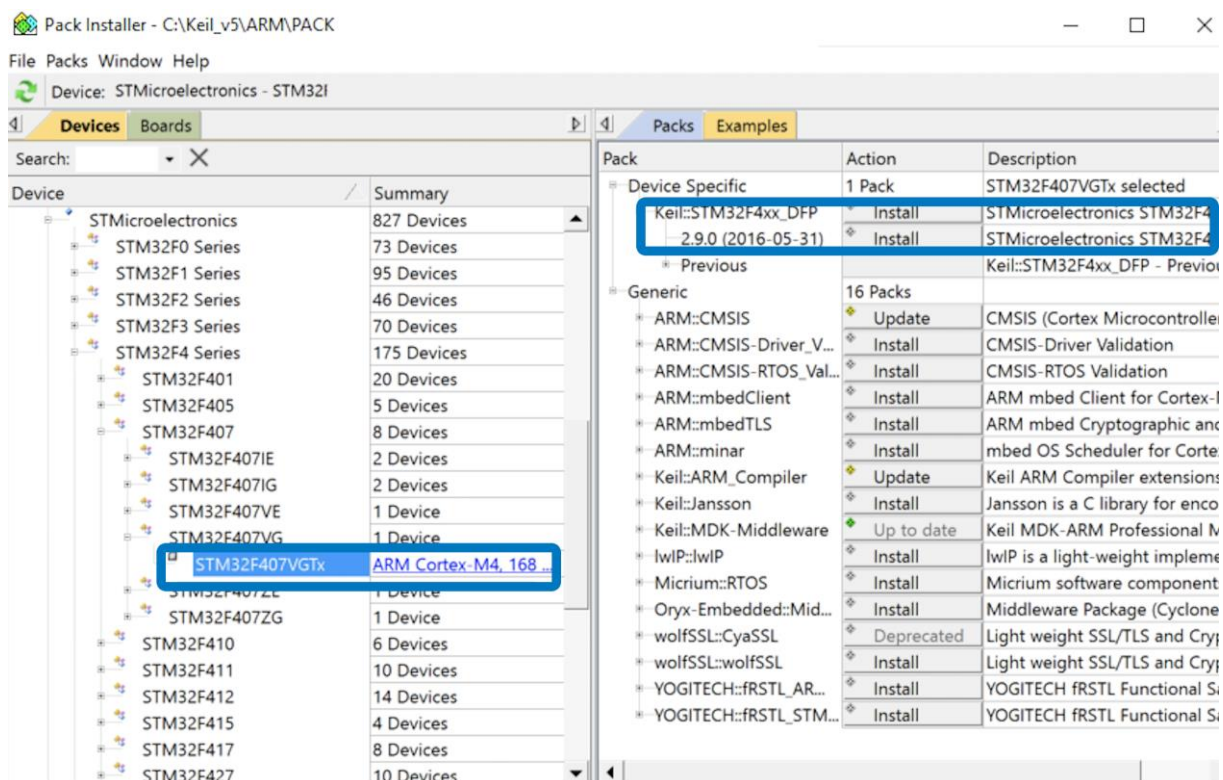
## 配置开发环境

### 配置 RTOS 开发环境

- 安装和相关开发工具
  - C Compiler: Armcc.exe V5.06 update 6 (build 750)
  - Assembler: Armasm.exe V5.06 update 6 (build 750)
  - Linker/Locator: ArmLink.exe V5.06 update 6 (build 750)
  - Library Manager: ArmAr.exe V5.06 update 6 (build 750)
  - Hex Converter: FromElf.exe V5.06 update 6 (build 750)
- 使用 Keil 软件

**激活 Keil MDK 软件后**，使用 Keil Pack Installer 或[手动下载](#)最新的 STM32F4xx\_DFP.2.x.x 驱动包，如 图 1.安装 Pack 包 所示。

图 1.安装 Pack 包



## 配置 Linux 开发环境

使用 Linux 开发环境时，请安装如下开发工具：

- C 编译器：GCC 5.4.0/5.5.0 版本
- CMake：2.8 及以上版本
- FFmpeg：4.1.3 及以上版本

## 运行示例程序

请下载 PSDK 提供的示例代码，通过编译、调试和烧录等操作获得示例程序。即可运行示例程序，借助示例程序了解使用 PSDK 开发负载设备的方法。



## 运行 RTOS 示例代码

1. **说明：** 本文以 **STM32F4 Discovery 开发板** 为例，介绍运行 RTOS 示例代码的步骤和方法。

## 烧录 Bootloader

使用 Keil MDK IDE 打开位于

sample/sample\_c/platform/rtos\_freertos/stm32f4\_discovery/project/mdk\_bootloader/  
目录下的工程文件

mdk\_bootloader.uvprojx

2. 使用 Keil MDK IDE 编译工程为示例程序。
3. 将编译后的示例程序**烧录**到负载设备中（如 STM32F4\_discovery 开发板）。

## 相关参考

- 实现 Bootloader：

platform/rtos\_freertos/stm32f4\_discovery/bootloader

Bootloader 工程目录：

platform/rtos\_freertos/stm32f4\_discovery/project/mdk\_bootloader

## 补充应用信息

使用 Keil IDE 打开位于 1.

1. sample/sample\_c/platform/rtos\_freertos/stm32f4\_discovery/project/mdk/ 目录下的工

程文件 `mdk_app.uvprojx`。

## 编译并烧录

- 使用 Keil MDK IDE 编译示例代码为示例程序。
- 编译示例代码后，将编译后的程序**烧录**到负载设备中（如 STM32F4\_discovery 开发板）。

如需调试示例程序，请将串口调试工具的波特率设置为：921600。

## 运行 Linux 示例代码

**说明：**介绍运行 Linux 示例代码的步骤和方法。

## 补充应用信息

在 `samples/sample_c/platform/linux/manifold2/application/GDU_sdk_app_info.h`

指定波特率。

```
#define USER_BAUD_RATE          "460800"
```

在 `samples/sample_c/platform/linux/manifold2/hal/hal_uart.h` 文件的 `LINUX_UART_DEV1`

和 `LINUX_UART_DEV2` 宏中填写对应的串口名称。

```
#define LINUX_UART_DEV1    "/dev/your_com"
```

```
#define LINUX_UART_DEV2    "/dev/your_com"
```

通过 `ifconfig` 命令，查看当前与无人机通讯的网口设备名称，并填写到

`samples/sample_c/platform/linux/manifold2/hal/hal_network.h` 文件

的 `LINUX_NETWORK_DEV` 宏中。

## 编译示例程序

编译示例代码

进入示例代码的目录：`sample/platform/linux/manifold2/project`，使用如下命令将示例代码编译为示例程序。

1.mkdir build

2.cd build

3.cmake ..

4.make

### 执行 C 语言示例程序

进入示例程序的目录：`cd build/bin/`

使用 `sudo ./GDU_sdk_demo_linux` 命令运行示例程序

### 执行 C++ 语言示例程序

进入示例程序的目录：`cd build/bin/`

使用 `sudo ./GDU_sdk_demo_linux_cxx` 命令运行示例程序

## 跨平台移植

- 将基于 PSDK 开发的负载设备控制程序移植到不同版本的软硬件平台上时，需要先初始化 Hal 和 Osal 层，注册关键的配置信息。通过加载静态库、引用指定的资源文件并声明结构体，设置负载设备控制程序跨平台移植所需的配置信息。最后使用指定的接口将 Platform 模块注册到负载设备的控制程序中，获取硬件资源和操作系统资源，实现负载设备控制程序的跨平台移植。

## 示例代码

- LinuxHal 层适配：`sample/platform/linux/manifold2/hal`

Osal 层适配：`sample/platform/linux/common/osal/`

FreeRTOSHal 层适配：

`sample/platform/rtos_freertos/psdk_development_board_1.0/hal`

Osal 层适配：`sample/platform/rtos_freertos/common/osal/`

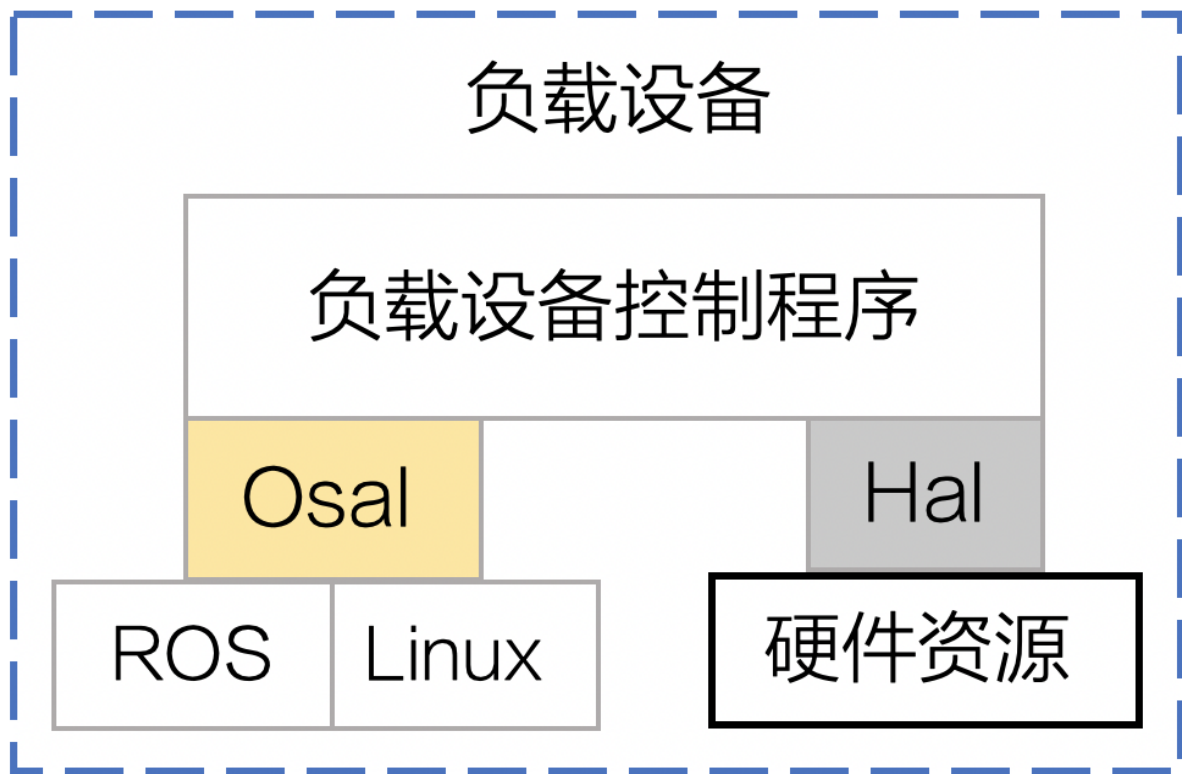
**说明：**PSDK Platform 模块的 API 接口，在

`psdk_lib/api_headers/psdk_platform.h` 文件中。

## 概述

为能使基于 PSDK 开发的负载设备控制程序移植到不同的软硬件平台，需要通过 Hal（Hardware Abstraction Layer，硬件接口层）适配不同的硬件平台，通过 Osal（Operating System Abstraction Layer，操作系统抽象层）实现与不同操作系统的兼容，如 图 1.代码移植 所示。

图 1.代码移植



## 基础概念

### Hal 层

Hal ( Hardware Abstraction Layer , 硬件接口层 ) 是 PSDK 硬件接口抽象层 , 位于操作系统、负载设备控制程序和硬件接口间。开发者需要按照

GduPlatform\_RegHalUartHandler() 与 GduPlatform\_RegHalNetworkHandler() 接口中的函数原型 , 实现并将适配 Hal 层的函数注册到负载设备控制程序中 , 使基于 PSDK 开发的负载设备控制程序 , 通过 Hal 层即可直接访问负载设备硬件的底层资源 , 控制负载设备执行相应的动作 , 使负载设备控制程序能够适配不同的硬件平台 , 如 STM32F407IGH6-EVAL 。

### 网口设备

需要使用网口的设备适配 Hal 层函数需要执行如下操作 :

- 实现适配 Hal 层网口操作函数本地网络配置：`T_GduReturnCode`

```
(*NetWorkConfig)(const char *ipAddr, const char *netMask)
```

使用 `GduPlatform_RegHalNetworkHandler()` 接口注册网口操作函数

## 串口设备

使用串口通信的设备适配 Hal 层函数需要执行如下操作：

实现适配 Hal 层 UART 操作函数串口初始化：`T_GduReturnCode (*UartInit)(void)`

发送数据：`T_GduReturnCode (*UartWriteData)(const uint8_t *buf, uint16_t len)`

接收数据：`T_GduReturnCode (*UartReadData)(uint8_t *buf, uint16_t len, uint16_t *realLen)`

使用 `GduPlatform_RegHalUartHandler()` 接口注册串口操作函数

## Osai 层

Osai ( Operating System Abstraction Layer , 操作系统抽象层 ) 是 PSDK 的操作系统抽象层，位于负载设备控制程序和操作系统间。开发者需要按照

`GduPlatform_RegOsaiHandler()` 接口中的函数原型，实现并将适配不同操作系统的函数注册到负载设备控制程序中，使用 PSDK 开发的负载设备控制程序即可直接访问操作系统以及操作系统内核的资源，将负载设备控制程序移植到不同的操作系统上。

## 线程函数

使用线程机制管理负载设备控制程序执行相应的任务，开发者需要实现创建线程、销毁线程和线程睡眠的函数。

- 创建线程：`T_GduReturnCode (*TaskCreate)(T_GduTaskHandle *task, void`

`*(taskFunc)(void *),uint32_t stackSize,void *arg)`

- 销毁线程：`T_GduReturnCode (*TaskDestroy)(T_GduTaskHandle task)`
- 线程睡眠：`T_GduReturnCode (*TaskSleepMs)(uint32_t timeMs)`

## 互斥锁

互斥锁是一种用于防止多个线程同时对同一队列、计数器和中断处理程序等公共资源（如共享内存等）执行读写操作的机制，能够有效避免进程死锁或长时间的等待。使用互斥锁机制，需要开发者实现创建互斥锁、销毁互斥锁、互斥锁上锁和互斥锁解锁。

- 创建互斥锁：`T_GduReturnCode (*MutexCreate)(T_GduMutexHandle *mutex)`
- 销毁互斥锁：`T_GduReturnCode (*MutexDestroy)(T_GduMutexHandle mutex)`
- 互斥锁上锁：`T_GduReturnCode (*MutexLock)(T_GduMutexHandle mutex)`
- 互斥锁解锁：`T_GduReturnCode (*MutexUnlock)(T_GduMutexHandle mutex)`

## 信号量

信号量是一种用于防止多线程同时操作相同代码段的机制。开发者使用该机制时，需要实现创建信号量、销毁信号量、等待信号量、释放信号量和等待超时信号量函数。

- 创建信号量：`T_GduReturnCode (*SemaphoreCreate)(T_GduSemHandle`

`*semaphore , uint32_t initValue)`

**说明：**使用该接口时，请设置 `initValue` 信号量的初始值。

- 销毁信号量：`T_GduReturnCode (*SemaphoreDestroy)(T_GduSemHandle`

semaphore)

- 等待信号量：T\_GduReturnCode (\*SemaphoreWait)(T\_GduSemHandle semaphore)

**说明：** 等待信号量接口等待时间的**最大值为 32767 ms**。

- 等待超时信号量：T\_GduReturnCode (\*SemaphoreTimedWait)(T\_GduSemHandle

semaphore , uint32\_t waitTimeMs)

- 释放信号量：T\_GduReturnCode (\*SemaphorePost)(T\_GduSemHandle semaphore)

## 时间接口

获取当前系统的时间 ( ms ) : T\_GduReturnCode (\*GetTimeMs)(uint32\_t \*ms)

## 内存管理接口

- 申请内存：void \*(\*Malloc)(uint32\_t size)
- 释放内存：void (\*Free)(void \*ptr)

## 实现跨平台移植

### 跨平台接口适配

## Hal 层适配

## 串口

- Linux 请根据硬件连接，配置对应的串口设备名称，如 ttyUSB0 并实现串口初始化、串口读数据和串口写数据的回调函数。 详细实现方法请参见：

sample/platform/linux/manifold2/hal/hal\_uart.c



- RTOS

请根据 MCU 的型号配置对应的串口管脚，并实现串口初始化、串口读数据和串口写数据的回调函数。详细实现方法请参见：

sample/platform/rtos\_freertos/stm32f4\_eval/hal/hal\_uart.c

## 网口

- Linux

使用网口将第三方开发平台连接至 GDU 无人机后，需要实现并注册配置网络的回调函数，当系统初始化时，会自动完成负载网络参数的配置，配置完成后，可以使用网口相关的功能。详细实现方法请参见：

/sample/platform/linux/manifold2/hal/hal\_network.c

- RTOS 对于 RTOS 系统，可以通过注册配置网络参数的回调函数，获取到当前负载应配置的网络参数，根据实际需要告知其他子系统模块，完成网口相关的功能。详细实现方法请参见：/sample/platform/rtos\_freertos/stm32f4\_eval/application/main.c

## Osai 层适配

- Linux

使用标准库 pthread 封装 T\_GduOsaiHandler 中的线程函数、互斥锁、信号量以及时间接口等接口。

详细实现方法请参见：sample/platform/linux/common/osai/osai.c

- RTOS

使用 CMSIS 封装的 thread 接口，封装 T\_GduOsaiHandler 中的线程函数、互斥锁、信号量以及时间接口等接口。

详细实现方法请参见：sample/platform/rtos\_freertos/common/osal/osal.c

## 注册跨平台适配接口

### 结构体声明

请完整地填充 T\_GduHalUartHandler、  
T\_GduHalNetWorkHandler 和 T\_GduOsalHandler  
中的接口内容，确保所注册的接口能够正常使用。

- T\_GduHalUartHandler halUartHandler

```
T_GduHalUartHandler halUartHandler = {  
    .UartInit = Hal_UartInit,  
    .UartReadData = Hal_UartReadData,  
    .UartWriteData = Hal_UartSendData,  
};
```

- T\_GduHalNetWorkHandler halNetWorkHandler

```
T_GduHalNetWorkHandler halNetWorkHandler = {  
    .NetWorkConfig = HalNetWork_Config,  
};
```

- T\_GduHalUartHandler osalHandler

```
T_GduOsalHandler osalHandler = {  
    .Malloc = Osal_Malloc,  
    .Free = Osal_Free,  
    .TaskCreate = Osal_TaskCreate,  
    .TaskDestroy = Osal_TaskDestroy,  
    .TaskSleepMs = Osal_TaskSleepMs,  
    .MutexCreate = Osal_MutexCreate,  
    .MutexDestroy = Osal_MutexDestroy,  
    .MutexLock = Osal_MutexLock,  
    .MutexUnlock = Osal_MutexUnlock,
```

```
.SemaphoreCreate = Osal_SemaphoreCreate,  
.SemaphoreDestroy = Osal_SemaphoreDestroy,  
.SemaphoreWait = Osal_SemaphoreWait,  
.SemaphorePost = Osal_SemaphorePost,  
.SemaphoreTimedWait = Osal_SemaphoreTimedWait,  
.GetTimeMs = Osal_GetTimeMs,  
};
```

请依次调用 GduPlatform\_RegHalUartHandler()、

GduPlatform\_RegHalNetworkHandler()和 GduPlatform\_RegOsalHandler()函数注册

Hal 层和 Osal 层，若接口注册不成功，请根据返回码和日志信息排查错误问题。

**说明：**跨平台移植模块必须要在其他 PSDK 功能模块前被注册，

若 Platform 模块注册失败或未注册，开发者将无法使用基于

PSDK 开发的负载设备。

```
if (GduPlatform_RegHalUartHandler(&halUartHandler) != GDU_RETURN_CODE_OK) {  
    printf("psdk register hal uart handler error");  
    return GDU_RETURN_CODE_ERR_UNKNOWN;  
}  
  
if (GduPlatform_RegHalNetworkHandler(&halNetWorkHandler) != GDU_RETURN_CODE_OK) {  
    printf("psdk register hal network handler error");  
    return GDU_RETURN_CODE_ERR_UNKNOWN;  
}  
  
if (GduPlatform_RegOsalHandler(&osalHandler) != GDU_RETURN_CODE_OK) {  
    printf("psdk register osal handler error");  
    return GDU_RETURN_CODE_ERR_UNKNOWN;  
}
```

# 功能合集

## 基础功能

### 日志管理

#### 概述

PSDK 的日志管理功能支持通过如串口、终端或 USB 等日志输出方法，输出 Debug、Info、Warn 和 Error 四种类型的日志；使用能够显示日志颜色的终端工具，如 Putty 等能够以不同的颜色显示不同类型的日志。

基于 PSDK 开发的负载设备输出的日志结构如下：日志颜色起始符 + 系统时间 + 模块名称 + 日志等级标识 + 日志内容 + 日志颜色结束符

#### 基础概念

### 日志标识符

使用**不支持**显示日志颜色的调试工具查看日志时，会显示日志的颜色标识符。

- 日志颜色：不同类型的日志，其标识颜色也不同。XShell、SecureCRT 及 Putty 等工具能够根据日志的等级，以不同的颜色显示不同类型的日志。
- 日志颜色起始标识符：
  - 黑色：\033[30m、红色：\033[31m、绿色：\033[32m
  - 黄色：\033[33m、蓝色：\033[34m、紫色：\033[35m
  - 青色：\033[36m、白色：\033[37m

- 日志颜色结束标识符：\033[0m

## 日志信息

- 系统时间：负载设备上电时，负载设备的时间为负载设备系统的时间，当负载设备与无人机完成时间同步后，负载设备的时间将与无人机的时间同步(ms)。
- 模块名称：PSDK 模块的名称（该名称无法被修改），用户打印接口的模块名称为“user”。
- 日志内容：单条日志最多不超过 500 个字节（bytes）。
- 日志等级：日志的等级从高到低为 Debug、Info、Warn 和 Error，日志管理功能模块可打印不高于指定等级的所有日志。表 1. 日志等级说明

日志等级	日志内容	输出接口	日志颜色
Debug - 4	调试信息	USER_LOG_DEBUG	White
Info - 3	关键信息	USER_LOG_INFO	Green
Warn - 2	警告信息	USER_LOG_WARN	Yellow
Error - 1	系统错误	USER_LOG_ERROR	Red

### 使用日志管理功能

使用 PSDK 日志管理功能，需要先初始化日志打印接口，设置日志等级，通过注册日志打印函数，实现日志管理功能。

**说明：**本教程以“使用 STM32 在 RTOS 系统上通过串口打印日志信息”为例，介绍使用日志管理功能的方法和步骤。

## 1. 初始化串口并注册日志输出方法

在 RTOS 系统上使用 PSDK 开发负载设备的日志管理功能时，建议使用串口打印日志信息。

```
static T_GduUser_PrintConsole(const uint8_t *data, uint16_t dataLen)
{
    UART_Write(GDU_CONSOLE_UART_NUM, (uint8_t *) data, dataLen);

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

```
UART_Init(GDU_CONSOLE_UART_NUM, GDU_CONSOLE_UART_BAUD);
```

## 2. 注册日志打印接口

使用 PSDK 的日志管理功能，需要初始化日志打印方法 printConsole，设置所需打印的日志等级、是否启动颜色显示和该日志对应的打印方法，并通过 GduLogger\_AddConsole()注册到负载设备控制程序中。

**注意：**使用 PSDK 开发的负载设备最多支持同时使用 **8 种** 日志打印方法。

```
T_GduLoggerConsole printConsole = {
    .func = GduUser_PrintConsole,
    .consoleLevel = GDU_LOGGER_CONSOLE_LOG_LEVEL_INFO,
    .isSupportColor = true,
};

returnCode = GduLogger_AddConsole(&printConsole);
if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    printf("add printf console error");
    goto out;
}
```

## 输出日志信息

调用日志打印函数输出日志信息，日志输出结果如 图 1.日志信息 所示。

```
USER_LOG_ERROR("psdk log console test.");  
USER_LOG_WARN("psdk log console test.");  
USER_LOG_INFO("psdk log console test.");  
USER_LOG_DEBUG("psdk log console test.");
```

**说明：** 日志的等级为 Info，因此负载设备中类型为 Info、Warn 及 Error 的日志将被输出在终端上，类型为 Debug 的日志将不会被打印出来。

## 日志查看

- Linux：

使用 grep 命令：在终端中执行./demo\_linux\_ubuntu | grep 'Info'过滤所需的日志信息，更多使用方法请使用 man grep 查询

- RTOS：使用串口查看工具如 SecureCRT、XShell 或 Putty 等工具查看日志信息

## 信息管理

### 概述

PSDK 的信息管理功能包含信息获取和消息订阅功能，基于 PSDK 开发的负载设备具

有信息获取功能，能够主动获取到无人机的型号、负载设备挂载的位置以及用户使用的移动端 App 等信息，加载不同的配置文件，方便用户使用负载设备；具有消息订阅功能的负载设备，能够记录用户订阅的数据信息，方便用户实现更广泛的应用。

## 基础概念

### 信息获取

信息获取是指负载设备能够**主动获取并记录**无人机上如无人机型号、硬件平台类型和负载设备挂载位置等数据信息。

**说明：**将使用 PSDK 开发的负载设备安装到无人机上，在开机初始化 5s 后才能够获取到无人机正确的数据信息。

使用 PSDK 开发的负载设备在初始化后，即可获取到如下信息：

- 基本信息：无人机型号、硬件平台类型和负载挂载位置
- 移动端 App 信息：App 的系统语言和 App 的屏幕类型

### 消息订阅

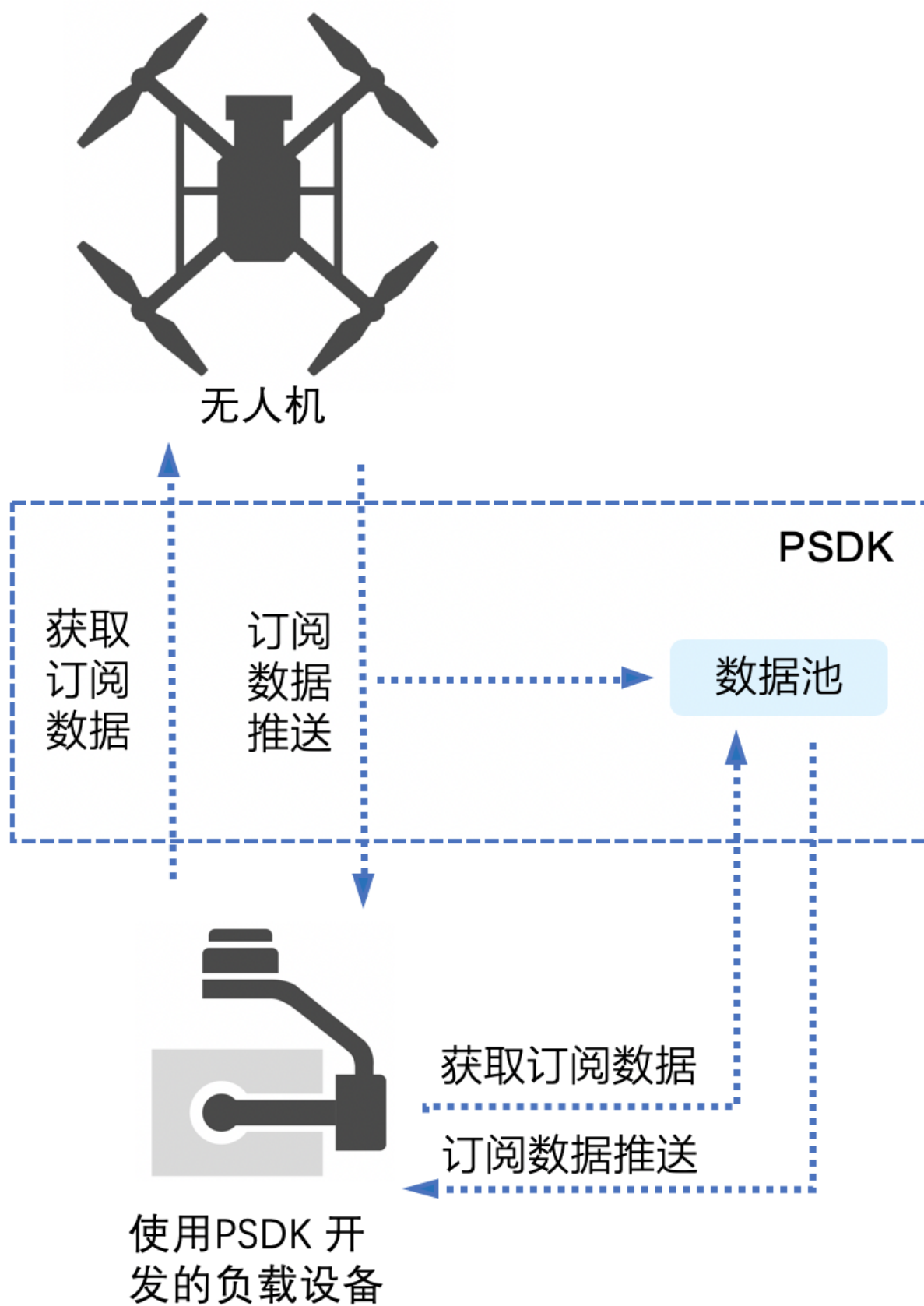
无人机上的各个部件根据无人机实际的飞行状况，会实时产生大量的数据信息并被无人机推送给其他模块，用户使用具有消息订阅功能的负载设备，能够指定所需订阅的数据信息。

### 订阅流程

订阅数据项后，负载设备即可获得订阅的信息，具体流程如 图 1.消息订阅 所示。



图 1.消息订阅



# 订阅项

使用 PSDK 消息订阅功能可订阅的数据信息如 表 1.无人机订阅项 所示。

表 1.无人机订阅项

数据类型	订阅项（topic）	最大订阅频率（Hz）
基础信息	速度	10
	融合海拔高度	10
	相对高度	10
	融合位置	10
	飞行状态	10
	电池信息	10
GPS 信息	GPS 日期	5
	GPS 时间	5
	GPS 位置	5
	GPS 速度	5
	GPS 信息	5
	GPS 信号等级	10
RTK 信息	RTK 位置	5
	RTK 速度	5
	RTK 航向角	5
	RTK 位置属性	5
	RTK 航向角属性	5

# 订阅规则

指定订阅频率时，任何参数的订阅频率不能小于或等于 0

## 使用消息订阅功能

- PSDK 支持通过注册回调和接口调用两种方式订阅无人机对外推送的数据信息：

通过调用 `GduFcSubscription_GetLatestValueOfTopic()` 获取无人机最新产生的订阅项的数据信息及其对应的时间。

- 通过调用 `GduFcSubscription_SubscribeTopic` 接口指定订阅频率和订阅项，通过构造并注册回调函数，获取无人机最新产生的订阅项的数据信息及其对应的时间。

**说明：** 使用订阅功能将接收到订阅项的数据与该数据产生时无人机系统的时间，该时间**暂不支持**与负载设备上的时间实现同步。

## 消息订阅功能模块初始化

使用 PSDK 开发的负载设备如需订阅无人机上的状态信息，需要先调用 `GduFcSubscription_Init()` 初始化消息订阅模块。

```
GduStat = GduFcSubscription_Init();
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("init data subscription module error.");
    return GDU_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
}
```

## 通过构造回调函数获取无人机上的信息

### 1. 构造回调函数

通过构造回调函数接收无人机推送的信息。

**注意：** 为避免出现内存踩踏事件，须将数据地址的类型强制转换为订阅项数据结构中的指针类型。

```

static T_GduTest_FcSubscriptionReceiveQuaternionCallback(const uint8_t *data, uint16_t dataSize, const
T_GDUDataTimestamp *timestamp)
{
    T_GduFcSubscriptionQuaternion *quaternion = (T_GduFcSubscriptionQuaternion *) data;
    GDU_f64_t pitch, yaw, roll;

    USER_UTIL_UNUSED(dataSize);

    pitch = (GDU_f64_t) asinf(-2 * quaternion->q1 * quaternion->q3 + 2 * quaternion->q0 *
quaternion->q2) * 57.3;
    roll = (GDU_f64_t) atan2f(2 * quaternion->q1 * quaternion->q2 + 2 * quaternion->q0 * quaternion->q3,
-2 * quaternion->q2 * quaternion->q2 - 2 * quaternion->q3 * quaternion->q3 + 1) * 57.3;
    yaw = (GDU_f64_t) atan2f(2 * quaternion->q2 * quaternion->q3 + 2 * quaternion->q0 *
quaternion->q1, -2 * quaternion->q1 * quaternion->q1 - 2 * quaternion->q2 * quaternion->q2 + 1) * 57.3;

    if (s_userFcSubscriptionDataShow == true) {
        USER_LOG_INFO("receive quaternion data.");

        USER_LOG_INFO("timestamp: millisecond %u microsecond %u.", timestamp->millisecond,
timestamp->microsecond);
        USER_LOG_INFO("quaternion: %f %f %f %f.\r\n", quaternion->q0, quaternion->q1,
quaternion->q2, quaternion->q3);
        USER_LOG_INFO("euler angles: pitch = %.2f roll = %.2f yaw = %.2f.", pitch, yaw, roll);
        GduTest_WidgetLogAppend("pitch = %.2f roll = %.2f yaw = %.2f.", pitch, yaw, roll);
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

## 1. 注册回调函数

注册回调函数接收无人机产生并对外推送的数据信息，下述代码以 1Hz 的频率订阅无人机“无人机飞行速度”和“无人机 GPS 坐标”，如图 2. 订阅结果（1）所示。

**说明：**使用订阅功能订阅无人机上的数据信息时，订阅频率只能为“最大订阅频率”的约数。

```
GduStat = GduFcSubscription_SubscribeTopic(GDU_FC_SUBSCRIPTION_TOPIC_VELOCITY,
```

```

GDU_DATA_SUBSCRIPTION_TOPIC_1_HZ,

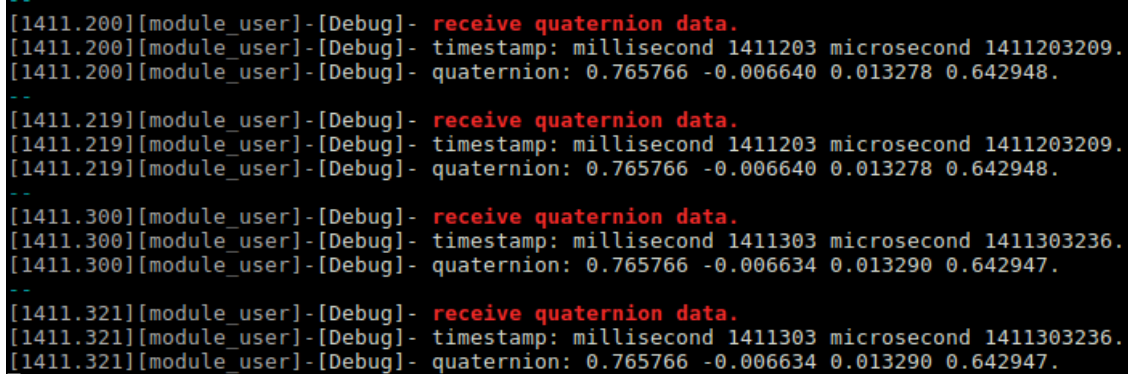
                                NULL);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Subscribe topic velocity error.");
    return GDU_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
}

GduStat      =      GduFcSubscription_SubscribeTopic(GDU_FC_SUBSCRIPTION_TOPIC_GPS_POSITION,
GDU_DATA_SUBSCRIPTION_TOPIC_1_HZ,

                                NULL);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Subscribe topic gps position error.");
    return GDU_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
}

```

图 2. 订阅结果 ( 1 )



```

[1411.200][module_user]-[Debug]- receive quaternion data.
[1411.200][module_user]-[Debug]- timestamp: millisecond 1411203 microsecond 1411203209.
[1411.200][module_user]-[Debug]- quaternion: 0.765766 -0.006640 0.013278 0.642948.
--
[1411.219][module_user]-[Debug]- receive quaternion data.
[1411.219][module_user]-[Debug]- timestamp: millisecond 1411203 microsecond 1411203209.
[1411.219][module_user]-[Debug]- quaternion: 0.765766 -0.006640 0.013278 0.642948.
--
[1411.300][module_user]-[Debug]- receive quaternion data.
[1411.300][module_user]-[Debug]- timestamp: millisecond 1411303 microsecond 1411303236.
[1411.300][module_user]-[Debug]- quaternion: 0.765766 -0.006634 0.013290 0.642947.
--
[1411.321][module_user]-[Debug]- receive quaternion data.
[1411.321][module_user]-[Debug]- timestamp: millisecond 1411303 microsecond 1411303236.
[1411.321][module_user]-[Debug]- quaternion: 0.765766 -0.006634 0.013290 0.642947.

```

## 在线程函数中获取无人机上的信息

通过数据订阅线程函数获取无人机推送的信息并打印在终端上。下述代码以 1Hz 的频率，订阅无人机最新产生的无人机飞行速度和无人机 GPS 坐标，以及该数据对应的时间，如 图 3. 订阅结果 ( 2 ) 所示。

```

GduStat = GduFcSubscription_GetLatestValueOfTopic(GDU_FC_SUBSCRIPTION_TOPIC_VELOCITY,
                                                    (uint8_t *) &velocity,
                                                    sizeof(T_GduFcSubscriptionVelocity),
                                                    &timestamp);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("get value of topic velocity error.");
}

```

```

} else {
    USER_LOG_INFO("velocity: x = %f y = %f z = %f healthFlag = %d.", velocity.data.x, velocity.data.y,
        velocity.data.z, velocity.health);
}

GduStat = GduFcSubscription_GetLatestValueOfTopic(GDU_FC_SUBSCRIPTION_TOPIC_GPS_POSITION,
    (uint8_t *) &gpsPosition,
    sizeof(T_GduFcSubscriptionGpsPosition),
    &timestamp);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("get value of topic gps position error.");
} else {
    USER_LOG_INFO("gps position: x = %d y = %d z = %d.", gpsPosition.x, gpsPosition.y, gpsPosition.z);
}

```

图 3. 订阅结果 ( 2 )

```

[1377.053][module_user]-[Debug]- timestamp: millisecond 1376958 microsecond 1376958197.
[1377.053][module_user]-[Debug]- quaternion: 0.765992 -0.006591 0.013253 0.642679.
[1377.053][module_user]-[Debug]- velocity: x 0.001853 y -0.000269 z -0.005251, healthFlag 1.
[1378.053][module_user]-[Debug]- timestamp: millisecond 1378058 microsecond 1378058200.
[1378.053][module_user]-[Debug]- quaternion: 0.765987 -0.006655 0.013201 0.642686.
[1378.053][module_user]-[Debug]- velocity: x 0.003893 y -0.002394 z -0.008591, healthFlag 1.
[1379.062][module_user]-[Debug]- timestamp: millisecond 1378958 microsecond 1378958211.
[1379.062][module_user]-[Debug]- quaternion: 0.766032 -0.006622 0.013278 0.642632.
[1379.063][module_user]-[Debug]- velocity: x -0.000899 y 0.001561 z -0.008762, healthFlag 1.
[1380.064][module_user]-[Debug]- timestamp: millisecond 1380058 microsecond 1380058204.
[1380.064][module_user]-[Debug]- quaternion: 0.766095 -0.006687 0.013258 0.642556.
[1380.064][module_user]-[Debug]- velocity: x -0.000179 y -0.003040 z -0.007479, healthFlag 1.

```

## 基础相机功能

2022-09-14 暂无评分 [Github Edit open in new window](#)

### 概述

为满足开发者对相机类负载设备的控制需求，PSDK 提供了**控制**相机执行拍照、录像、等功能的接口，开发者需**先实现**相机拍照、录像以及测光等功能，再通过注册 PSDK 相机类的接口，开发出功能完善的相机类负载设备；通过使用 GduFlight2 以及基于 MSDK 开发的移动端 App，用户能够控制使用 PSDK 开发的相机类负载设备执行指定的动作，获取负载设备中的信息和资源。

- 基础功能：设置相机模式、拍照、录像、获取相机状态、SD 卡管理

## 基础概念介绍

## 相机模式

使用相机类功能前，需要先设置相机类负载设备的模式，不同的模式指定了相机类负载设备在执行某个任务时的工作逻辑。

- 拍照：在该模式下，用户能够触发相机类负载设备拍摄照片。
- 录像：在该模式下，用户能够触发相机类负载设备录制影像。

**注意：**相机只能在一种模式中执行相应的操作，如在录像模式下仅能录像无法拍照。

## 拍照模式

使用 PSDK 开发的相机类负载设备支持以下拍照模式：

- 单拍：下发拍照命令后，相机拍摄单张照片。

## 实现相机类基础功能

请开发者根据选用的**开发平台**以及行业应用实际的使用需求，按照 PSDK 中的结构体

T\_GduCameraCommonHandler

构造实现相机类负载设备设置相机模式、拍照和录像等功能的函数，将相机功能的函数注册到 PSDK 中指定的接口后，用户通过使用 GduFlight2 或基于 MSDK 开发的移动端 App 能够控制基于 PSDK 开发的相机类负载设备执行相应的动作。

```

// 获取负载设备系统当前的状态
s_commonHandler.GetSystemState = GetSystemState;
// 实现设置相机类负载设备模式的功能
s_commonHandler.SetMode = SetMode;
s_commonHandler.GetMode = GduTest_CameraGetMode;
// 实现开始或停止录像的功能
s_commonHandler.StartRecordVideo = StartRecordVideo;
s_commonHandler.StopRecordVideo = StopRecordVideo;
// 实现开始或停止拍照的功能
s_commonHandler.StartShootPhoto = StartShootPhoto;
s_commonHandler.StopShootPhoto = StopShootPhoto;
// 实现设置相机类负载设备的拍照功能
s_commonHandler.SetShootPhotoMode = SetShootPhotoMode;
s_commonHandler.GetShootPhotoMode = GetShootPhotoMode;
s_commonHandler.SetPhotoBurstCount = SetPhotoBurstCount;
s_commonHandler.GetPhotoBurstCount = GetPhotoBurstCount;
s_commonHandler.SetPhotoTimeIntervalSettings = SetPhotoTimeIntervalSettings;
s_commonHandler.GetPhotoTimeIntervalSettings = GetPhotoTimeIntervalSettings;
// 实现 SD 卡管理功能
s_commonHandler.GetSDCardState = GetSDCardState;
s_commonHandler.FormatSDCard = FormatSDCard;

```

## 基础功能初始化

使用 PSDK 开发负载设备的相机功能时，必须要初始化相机模块并注册相机类的功能。

## 相机类功能模块初始化

在使用相机类功能前，必须先调用接口 `GduPayloadCamera_Init` 初始化相机类负载设备，确保相机类负载设备可正常工作。

```

T_GduReturnCode returnCode;

returnCode = GduPayloadCamera_Init();
if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("payload camera init error:0x%08lX", returnCode);
}

```



## 注册相机类基础功能

开发者**实现**相机类负载设备设置相机模式、拍照和录像等功能后，需要通过

GduPayloadCamera\_RegCommonHandler 注册相机类基础功能。

```
returnCode = GduPayloadCamera_RegCommonHandler(&s_commonHandler);
if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("camera register common handler error:0x%08lX", returnCode);
}
```

### 使用 SD 卡管理功能

## 说明

- 本教程以**模拟** SD 卡管理功能为例，介绍使用 PSDK SD 卡管理功能的使用方法，如需开发具有 SD 卡管理功能的负载设备，请调用负载设备系统的接口实现 SD 卡管理功能。

## SD 卡模块初始化

使用 SD 卡管理功能，需要开发者先开发并注册操作 SD 卡功能的函数，通过初始化

SD 卡管理模块，获取 SD 卡的状态信息。

```
s_cameraSDCardState.isInserted = true;
s_cameraSDCardState.totalSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
s_cameraSDCardState.remainSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
s_cameraSDCardState.availableCaptureCount      =      SDCARD_TOTAL_SPACE_IN_MB      /
SDCARD_PER_PHOTO_SPACE_IN_MB;
s_cameraSDCardState.availableRecordingTimeInSeconds =      SDCARD_TOTAL_SPACE_IN_MB      /
SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;
```

# 获取 SD 卡的当前状态

基于 PSDK 开发的负载设备控制程序调用

GetSDCardState

接口能够获取负载设备上 SD 卡当前的状态，用户使用 GduFlight2 以及基于 MSDK

开发的 APP 能够查看负载设备中 SD 卡的状态信息。

```
// 预估可拍照张数和可录像时长的功能。
if (s_cameraState.isRecording) {
    s_cameraState.currentVideoRecordingTimeInSeconds++;
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;
    if (s_cameraSDCardState.remainSpaceInMB > SDCARD_TOTAL_SPACE_IN_MB) {
        s_cameraSDCardState.remainSpaceInMB = 0;
        s_cameraSDCardState.isFull = true;
    }
}
// 获取 SD 卡的状态
static T_GduReturnCode GetSDCardState(T_GDUCameraSDCardState *sdCardState)
{
    T_GduReturnCode returnCode;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    memcpy(sdCardState, &s_cameraSDCardState, sizeof(T_GDUCameraSDCardState));

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

## 使用 SD 卡格式化功能

基于 PSDK 开发的负载设备控制程序调用 FormatSDCard 接口能够控制负载设备执行 SD 卡格式化，用户使用 GduFlight2 以及基于 MSDK 开发的 APP 可获取负载设备中 SD 卡的状态信息并控制负载设备执行 SD 卡格式化功能，如图 1. SD 卡管理功能所示。

```
static T_GduReturnCode FormatSDCard(void)
{
    T_GduReturnCode returnCode;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("format sdcard");

    memset(&s_cameraSDCardState, 0, sizeof(T_GDUCameraSDCardState));
    s_cameraSDCardState.isInserted = true;
    s_cameraSDCardState.totalSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
    s_cameraSDCardState.remainSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
    s_cameraSDCardState.availableCaptureCount = SDCARD_TOTAL_SPACE_IN_MB /
    SDCARD_PER_PHOTO_SPACE_IN_MB;
    s_cameraSDCardState.availableRecordingTimeInSeconds =
        SDCARD_TOTAL_SPACE_IN_MB / SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

## 使用模式设置功能

基于 PSDK 开发的负载设备控制程序调用 SetMode 和 GetMode 接口能够设置相机的模式，用户使用 GduFlight2 能够切换相机类负载设备的工作模式，如 图 2. 设置相机模式 所示。

```
static T_GduReturnCode GetSystemState(T_GDUCameraSystemState *systemState)
{
    T_GduReturnCode returnCode;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *systemState = s_cameraState;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_GduReturnCode SetMode(E_GDUCameraMode mode)
{
    T_GduReturnCode returnCode;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_cameraState.cameraMode = mode;
    USER_LOG_INFO("set camera mode:%d", mode);

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
```

```

        if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
            USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
            return returnCode;
        }

        return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
    }

T_GduReturnCode GduTest_CameraGetMode(E_GDUCameraMode *mode)
{
    T_GduReturnCode returnCode;
    T_GduOsalHandler *osalHandler = GduPlatform_GetOsalHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *mode = s_cameraState.cameraMode;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

## 使用拍照功能

### 说明

- 使用拍照功能前，用户需要在 GduFlight2 或基于 MSDK 开发的移动端 App 上将相机类负载设备的工作模式设置为拍照模式。

- 使用 PSDK 开发的负载设备在拍照时，会向 GduFlight2 或基于 MSDK 开发的移动端 App 返回拍照状态（用于如触发移动端 App 拍照声音等功能）。

## 设置相机类负载设备的拍照模式

基于 PSDK 开发的负载设备控制程序调用 SetShootPhotoMode 和 GetShootPhotoMode 接口能够设置并获取相机类负载设备的模式，用户使用 GduFlight2 以及基于 MSDK 开发的移动端 App 可设置并获取相机类负载设备的拍照模式。

```
static T_GduReturnCode SetShootPhotoMode(E_GDUCameraShootPhotoMode mode)
{
    T_GduReturnCode returnCode;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_cameraShootPhotoMode = mode;
    USER_LOG_INFO("set shoot photo mode:%d", mode);

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

```
static T_GduReturnCode GetShootPhotoMode(E_GDUCameraShootPhotoMode *mode)
```

```

{
    T_GduReturnCode returnCode;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *mode = s_cameraShootPhotoMode;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);\
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

## 控制相机单拍

基于 PSDK 开发的负载设备控制程序调用 StartShootPhoto 和 StopShootPhoto

接口控制相机类负载设备拍摄单张照片，用户使用 GduFlight2 以及基于 MSDK 开发

的移动端 App 可控制相机类负载设备拍摄单张照片。

```

static T_GduReturnCode StartShootPhoto(void)
{
    T_GduReturnCode returnCode;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("start shoot photo");
    s_cameraState.isStoring = true;

    if (s_cameraShootPhotoMode == GDU_CAMERA_SHOOT_PHOTO_MODE_SINGLE) {

```

```

        s_cameraState.shootingState = GDU_CAMERA_SHOOTING_SINGLE_PHOTO;
    } else if (s_cameraShootPhotoMode == GDU_CAMERA_SHOOT_PHOTO_MODE_BURST) {
        s_cameraState.shootingState = GDU_CAMERA_SHOOTING_BURST_PHOTO;
    } else if (s_cameraShootPhotoMode == GDU_CAMERA_SHOOT_PHOTO_MODE_INTERVAL) {
        s_cameraState.shootingState = GDU_CAMERA_SHOOTING_INTERVAL_PHOTO;
        s_cameraState.isShootingIntervalStart = true;
        s_cameraState.currentPhotoShootingIntervalTimeInSeconds =
s_cameraPhotoTimeIntervalSettings.timeIntervalSeconds;
    }

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_GduReturnCode StopShootPhoto(void)
{
    T_GduReturnCode returnCode;
    T_GduOsalHandler *osalHandler = GduPlatform_GetOsalHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("stop shoot photo");
    s_cameraState.shootingState = GDU_CAMERA_SHOOTING_PHOTO_IDLE;
    s_cameraState.isStoring = false;
    s_cameraState.isShootingIntervalStart = false;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```



## 拍照状态管理

在 GduFlight2 以及使用 MSDK 开发的移动端 App 中点击“拍照”按钮后，使用 PSDK 开发的相机类负载设备在自定义时间内（如 0.5s）在线程中执行拍照、照片存储和内存状态更新的操作。

## 确认拍照状态

使用 PSDK 开发的相机类负载设备在执行完拍照动作后，需要获取负载设备的拍照状态。

```
if (s_cameraState.shootingState != PSDK_CAMERA_SHOOTING_PHOTO_IDLE &&
    photoCnt++ > TAKING_PHOTO_SPENT_TIME_MS_EMU / (1000 /
PAYLOAD_CAMERA_EMU_TASK_FREQ)) {
    s_cameraState.isStoring = false;
    s_cameraState.shootingState = PSDK_CAMERA_SHOOTING_PHOTO_IDLE;
    photoCnt = 0;
}
```

## 存储照片

相机类负载设备在执行完拍照后，使用 PSDK 开发的相机类负载设备将相机拍摄的照片存储在**相机类负载设备**上的内存卡中。

- 存储单拍模式下相机类负载设备拍摄的照片

```
if (s_cameraShootPhotoMode == GDU_CAMERA_SHOOT_PHOTO_MODE_SINGLE) {
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB;
    s_cameraState.isStoring = false;
    s_cameraState.shootingState = GDU_CAMERA_SHOOTING_PHOTO_IDLE;
}
```

- 存储连拍模式下相机类负载设备拍摄的照片

```
else if (s_cameraShootPhotoMode == GDU_CAMERA_SHOOT_PHOTO_MODE_BURST) {
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB *
s_cameraBurstCount;
    s_cameraState.isStoring = false;
    s_cameraState.shootingState = GDU_CAMERA_SHOOTING_PHOTO_IDLE;
}
```

- 存储定时拍照模式下相机类负载设备拍摄的照片

```
else if (s_cameraShootPhotoMode == GDU_CAMERA_SHOOT_PHOTO_MODE_INTERVAL) {
    if (isStartIntervalPhotoAction == true) {
        s_cameraState.isStoring = false;
        s_cameraState.shootingState = GDU_CAMERA_SHOOTING_PHOTO_IDLE;
        s_cameraSDCardState.remainSpaceInMB =
            s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB;
    }
}
```

## 检查存储空间

为确保相机类负载设备中的 SD 卡在相机类负载设备执行拍照动作后，有充足的存储空间存储照片或视频，建议在使用 PSDK 开发的相机类负载设备中添加检查 SD 卡存储空间的功能。

- 检查相机类负载设备执行单拍和连拍后 SD 卡剩余的存储空间。

```
if (s_cameraSDCardState.remainSpaceInMB > SDCARD_TOTAL_SPACE_IN_MB) {
    s_cameraSDCardState.remainSpaceInMB = 0;
    s_cameraSDCardState.isFull = true;
}
```

- 检查相机类负载设备执行定时拍照后 SD 卡剩余的存储空间

```
if (s_cameraShootPhotoMode == GDU_CAMERA_SHOOT_PHOTO_MODE_INTERVAL) {
    if (isStartIntervalPhotoAction == true) {
        s_cameraState.isStoring = false;
    }
}
```

```
s_cameraState.shootingState = GDU_CAMERA_SHOOTING_PHOTO_IDLE;
s_cameraSDCardState.remainSpaceInMB =
    s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB;
}
}
```

## 使用录像功能

### 说明

- 相机类负载设备在录像的过程中无法拍照和测光；
- 开发者可根据用户的使用需要，设置相机类负载设备录像时如 ISO、曝光以及对焦等参数的默认值；
- 使用相机类负载设备的录像功能前，用户需要在 GduFlight2 或基于 MSDK 开发的移动端 App 上将相机类负载设备的模式设置为录像模式。

### 控制相机录像

基于 PSDK 开发的负载设备控制程序调用 StartRecordVideo 和 StopRecordVideo 接口控制相机类负载设备录像，用户使用 GduFlight2 以及基于 MSDK 开发的移动端 App 可控制相机类负载设备录像。

```

static T_GduReturnCode StartRecordVideo(void)
{
    T_GduReturnCode GduStat;
    T_GduReturnCode returnCode = GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    GduStat = osalHandler->MutexLock(s_commonMutex);
    if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", GduStat);
        return GduStat;
    }

    if (s_cameraState.isRecording != false) {
        USER_LOG_ERROR("camera is already in recording state");
        returnCode = GDU_ERROR_SYSTEM_MODULE_CODE_NONSUPPORT_IN_CURRENT_STATE;
        goto out;
    }

    s_cameraState.isRecording = true;
    USER_LOG_INFO("start record video");

out:
    GduStat = osalHandler->MutexUnlock(s_commonMutex);
    if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", GduStat);
        return GduStat;
    }

    return returnCode;
}

```

```

static T_GduReturnCode StopRecordVideo(void)
{
    T_GduReturnCode GduStat;
    T_GduReturnCode returnCode = GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    GduStat = osalHandler->MutexLock(s_commonMutex);
    if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", GduStat);
        return GduStat;
    }

    if (s_cameraState.isRecording != true) {
        USER_LOG_ERROR("camera is not in recording state");
    }
}

```

```

        returnCode = GDU_ERROR_SYSTEM_MODULE_CODE_NONSUPPORT_IN_CURRENT_STATE;
        goto out;
    }

    s_cameraState.isRecording = false;
    s_cameraState.currentVideoRecordingTimeInSeconds = 0;
    USER_LOG_INFO("stop record video");

out:
    GduStat =osalHandler->MutexUnlock(s_commonMutex);
    if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", GduStat);
        return GduStat;
    }

    return returnCode;
}

```

## 录像状态更新

使用 PSDK 开发的相机类负载设备控制程序，默认以 10Hz 的频率更新相机的状态。

**说明：** 相机开始录像后，GduFlight2 及基于 MSDK 开发的移动端

App 会显示当前正在录像的时间，相机停止录像时，该时间将归 0。

```

if (s_cameraState.isRecording) {
    s_cameraState.currentVideoRecordingTimeInSeconds++;
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;
    if (s_cameraSDCardState.remainSpaceInMB > SDCARD_TOTAL_SPACE_IN_MB) {
        s_cameraSDCardState.remainSpaceInMB = 0;
        s_cameraSDCardState.isFull = true;
    }
}

static T_GduReturnCode GetSystemState(T_GDUCameraSystemState *systemState)
{
    T_GduReturnCode returnCode;

```

```

T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

returnCode = osalHandler->MutexLock(s_commonMutex);
if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
    return returnCode;
}

*systemState = s_cameraState;

returnCode = osalHandler->MutexUnlock(s_commonMutex);
if (returnCode != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
    return returnCode;
}

return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

在 GduFlight2 或基于 MSDK 开发的移动端 App 上向负载设备发送录像指令后（也可通过遥控器向负载设备发送录像指令），相机类负载设备根据用户发送的指令控制负载设备录像。

## 云台功能

## 说明

### 概述

使用 PSDK 的“云台控制”功能，开发者需要先设计负载设备的云台并开发出控制云台的程序，将云台的控制函数注册到 PSDK 指定的接口后，用户通过使用

GduFlight2、基于 MSDK 开发的移动端 App 及遥控器即可控制基于 PSDK 开发的具有云台功能的负载设备，同时获得负载设备的相关信息，如姿态等。

## 基础概念

### 云台状态信息

使用 PSDK 开发的云台类负载设备需要按照指定的要求上报云台的状态、当前姿态和校准状态等信息，方便用户移动端 App 或机载计算机根据云台的状态，实现精准控制。有关获取云台状态的方法和相关详情请参见 **PSDK API 文档**

## 关节角与姿态角

### 云台关节与云台关节角

云台关节是云台上带动负载设备转动的结构件：云台电机，云台关节角即云台电机转动的角度。本教程使用机体坐标系描述云台的关节角。

### 云台姿态与云台姿态角

云台的姿态如根据用户的控制指令，云台能够调整姿态；云台姿态角即使用大地坐标系（NED，北东地坐标系）描述云台上**负载设备**的角度，该角度也称为欧拉角。

# 云台控制

## 控制方式

- 速度控制：用户可控制使用 PSDK 开发的云台的转动速度。

## 说明

- 在速度控制模式下，云台根据用户指定的速度转动 0.5s，当云台转动到限位角时，将会停止转动。
- 当前只支持速度控制模式

## 实现云台功能

请开发者根据选用的**开发平台**以及行业应用实际的使用需求，按照 PSDK 中的结构体 T\_GDUGimbalCommonHandler 构造实现云台类负载设备控制功能的函数，将云台控制功能的函数注册到 PSDK 中指定的接口后，用户通过使用 GduFlight2 或基于 MSDK 开发的移动端 App 能够控制基于 PSDK 开发的云台类负载设备执行指定的动作。

```
s_commonHandler.GetSystemState = GetSystemState;
s_commonHandler.GetAttitudeInformation = GetAttitudeInformation;
s_commonHandler.GetCalibrationState = GetCalibrationState;
s_commonHandler.GetRotationSpeed = GetRotationSpeed;
s_commonHandler.GetJointAngle = GetJointAngle;

s_commonHandler.Rotate = GduTest_GimbalRotate;
s_commonHandler.StartCalibrate = StartCalibrate;
```



```
s_commonHandler.SetControllerSmoothFactor = SetControllerSmoothFactor;  
s_commonHandler.SetPitchRangeExtensionEnabled = SetPitchRangeExtensionEnabled;  
s_commonHandler.SetControllerMaxSpeedPercentage = SetControllerMaxSpeedPercentage;  
s_commonHandler.RestoreFactorySettings = RestoreFactorySettings;  
s_commonHandler.SetMode = SetMode;  
s_commonHandler.Reset = Reset;  
s_commonHandler.FineTuneAngle = FineTuneAngle;
```

## 使用云台控制功能

使用云台控制功能，需要先实现云台控制功能，再实现云台限位功能，根据云台模式

调整云台的姿态、目标角度和限位标志，最后实现云台校准功能校准云台。

# 使用云台功能

## 1. 云台控制功能模块初始化

使用“云台控制”功能前，需要先初始化云台控制功能模块，确保云台控制功能可正常运行。

```
GduStat = GDUGimbal_Init();  
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {  
    USER_LOG_ERROR("init gimbal module error: 0x%08lX", GduStat);  
}
```

## 2.注册云台控制功能

使用 PSDK 的云台控制功能控制云台类负载设备时，开发者需要将控制云台的函数注册到指定的接口中。

```
GduStat = GDUGimbal_RegCommonHandler(&s_commonHandler);  
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {  
    USER_LOG_ERROR("gimbal register common handler error: 0x%08lX", GduStat);  
}
```

### 3. 获取云台的状态信息

为方便用户控制云台执行相应的动作，需调用 GetSystemState 接口获取云台的状态。

```
static T_GduReturnCode GetSystemState(T_GDUGimbalSystemState *systemState)
{
    T_GduOsHandler *osalHandler = GduPlatform_GetOsHandler();

    if (osalHandler->MutexLock(s_commonMutex) != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("mutex lock error");
        return GDU_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
    }

    *systemState = s_systemState;

    if (osalHandler->MutexUnlock(s_commonMutex) != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS)
    {
        USER_LOG_ERROR("mutex unlock error");
        return GDU_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
    }

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

### 4. 控制云台转动

负载设备根据云台的姿态和转动速度，将相对角度控制量、绝对角度控制量或速度控制量转换为控制云台转动的速度，根据该速度控制云台转动

```
nextAttitude.pitch =
    (float) s_attitudeHighPrecision.pitch + (float) s_speed.pitch / (float)
PAYLOAD_GIMBAL_TASK_FREQ;
nextAttitude.roll =
    (float) s_attitudeHighPrecision.roll + (float) s_speed.roll / (float)
PAYLOAD_GIMBAL_TASK_FREQ;
nextAttitude.yaw = (float) s_attitudeHighPrecision.yaw + (float) s_speed.yaw / (float)
PAYLOAD_GIMBAL_TASK_FREQ;

if (s_controlType == TEST_GIMBAL_CONTROL_TYPE_ANGLE) {
    nextAttitude.pitch =
```

```

        (nextAttitude.pitch - s_targetAttitude.pitch) * s_speed.pitch >= 0 ?
s_targetAttitude.pitch
        :
nextAttitude.pitch;
        nextAttitude.roll = (nextAttitude.roll - s_targetAttitude.roll) * s_speed.roll >= 0 ?
s_targetAttitude.roll
        : nextAttitude.roll;
        nextAttitude.yaw =
        (nextAttitude.yaw - s_targetAttitude.yaw) * s_speed.yaw >= 0 ? s_targetAttitude.yaw :
nextAttitude.yaw;
    }

    GduTest_GimbalAngleLegalization(&nextAttitude, s_aircraftAttitude,
&s_attitudeInformation.reachLimitFlag);
    s_attitudeInformation.attitude.pitch = nextAttitude.pitch;
    s_attitudeInformation.attitude.roll = nextAttitude.roll;
    s_attitudeInformation.attitude.yaw = nextAttitude.yaw;

    s_attitudeHighPrecision.pitch = nextAttitude.pitch;
    s_attitudeHighPrecision.roll = nextAttitude.roll;
    s_attitudeHighPrecision.yaw = nextAttitude.yaw;

    if (s_controlType == TEST_GIMBAL_CONTROL_TYPE_ANGLE) {
        if (memcmp(&s_attitudeInformation.attitude, &s_targetAttitude, sizeof(T_GDUAttitude3d))
== 0) {
            s_rotatingFlag = false;
        }
    } else if (s_controlType == TEST_GIMBAL_CONTROL_TYPE_SPEED) {
        if ((s_attitudeInformation.reachLimitFlag.pitch == true || s_speed.pitch == 0) &&
(s_attitudeInformation.reachLimitFlag.roll == true || s_speed.roll == 0) &&
(s_attitudeInformation.reachLimitFlag.yaw == true || s_speed.yaw == 0)) {
            s_rotatingFlag = false;
        }
    }
}

```

## 时间同步

### 概述

时间同步是一个用于同步负载设备时间和无人机时间的功能，PSDK 通过 PPS 信号（周期性的脉冲）同步负载设备和**具有 RTK 功能的无人机**的时间。具有“时间同步”功能的负载设备，能够方便用户顺利地使用日志排查无人机飞行过程中的各类故障、分析传感器采样的数据以及获取精准的定位信息等功能。

### 本文所指

- 无人机时间：无人机系统的时间。
- 本地时间：负载设备上的时间。

### 时间同步

安装在无人机上的使用 PSDK 开发的负载设备在上电后，将初始化时间同步功能模块，消除负载设备和无人机的时钟差，同步负载设备和无人机的时间。

**说明：**使用时间同步功能前，请通过移动端 App 确认无人机与 RTK 卫星间保持良好的通信状态；该移动端 App 可为 GDU 发布的 App，如 GduFlight2，也可为基于 MSDK 开发的移动端 App

1. 将负载设备安装在无人机的云台上，在无人机上电后，使用 PSDK 开发的负载设备将接收到无人机发送的 PPS 硬件脉冲信号；
2. 当负载设备检测到 PPS 信号的上升沿时，负载设备需要记录负载设备上的本地时间；
3. PSDK 的底层处理程序将获取与 PPS 信号同步的无人机系统上的时间。

## 注意

- a. 请确保 PPS 信号上升沿到达负载至负载记录本地时间之间的延迟低于 1ms。
4. 请使用硬件中断的形式实现 PPS 信号的响应。
  5. PSDK 的底层处理程序将计算负载设备本地的时间与无人机系统时间的时钟差，实现负载设备与无人机系统时间的同步。

负载设备通过 GDUTimeSync\_TransferToAircraftTime 接口将负载设备本地的时间转换为无人机系统上的时间。

## 使用时间同步功能

### 1. 配置 PPS 引脚参数

为了使 PPS 引脚能够正确接收 PPS 信号，需要设置 PPS 引脚的各项参数，并开启 PPS 硬件引脚的功能，接收 PPS 时间同步信号。

```

T_GduReturnCode GduTest_PpsSignalResponseInit(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /* Enable GPIOD clock */
    __HAL_RCC_GPIOD_CLK_ENABLE();

    /* Configure pin as input floating */
    GPIO_InitStructure.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStructure.Pull = GPIO_NOPULL;
    GPIO_InitStructure.Pin = PPS_PIN;
    HAL_GPIO_Init(PPS_PORT, &GPIO_InitStructure);

    /* Enable and set EXTI Line Interrupt to the lowest priority */
    HAL_NVIC_SetPriority(PPS_IRQn, PPS_IRQ_PRIO_PRE, PPS_IRQ_PRIO_SUB);
    HAL_NVIC_EnableIRQ(PPS_IRQn);

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

## 2. 时间同步功能模块初始化

使用时间同步功能，需要初始化时间同步模块。

```

GduStat = GDUTimeSync_Init();
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("time synchronization module init error.");
    return GduStat;
}

```

## 3. 开发并注册获取负载设备上本地时间的函数

基于 PSDK 开发的负载设备在使用时间同步功能时，需要以硬件中断的方式响应无人机推送的 PPS 时间同步信号；当最新的 PPS 信号被触发时，基于 PSDK 开发的负载设备控制程序，需要获取到负载设备本地的时间。

### 1. 开发并注册硬件中断处理函数

开发者需要实现硬件中断处理功能，并将处理硬件中断的函数注册到指定的接口中，当负载设备接收到 PPS 信号的上升沿时，硬件中断处理函数能够处理 PPS 时间同步信号。

```
void GduTest_PpsIrqHandler(void)
{
    T_GduReturnCode psdkStat;
    uint32_t timeMs = 0;

    /* EXTI line interrupt detected */
    if (__HAL_GPIO_EXTI_GET_IT(PPS_PIN) != RESET) {
        __HAL_GPIO_EXTI_CLEAR_IT(PPS_PIN);
        psdkStat = Osa1_GetTimeMs(&timeMs);
        if (psdkStat == GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS)
            s_ppsNewestTriggerLocalTimeMs = timeMs;
    }
}
```

## 1. 获取 PPS 信号被触发时负载设备的本地时间

开发者需要实现获取 PPS 信号被触发时负载设备上本地时间的功能，并将该功能的函数注册到指定的接口中。当负载设备接收到 PPS 信号的上升沿时，基于 PSDK 开发的负载设备控制程序能够记录 PPS 信号被触发时负载设备上的本地时间。

```
T_GduReturnCode GduTest_GetNewestPpsTriggerLocalTimeUs(uint64_t *localTimeUs)
{
    if (localTimeUs == NULL) {
        USER_LOG_ERROR("input pointer is null.");
        return GDU_ERROR_SYSTEM_MODULE_CODE_INVALID_PARAMETER;
    }

    if (s_ppsNewestTriggerLocalTimeMs == 0) {
        USER_LOG_WARN("pps have not been triggered.");
        return GDU_ERROR_SYSTEM_MODULE_CODE_BUSY;
    }

    *localTimeUs = (uint64_t) (s_ppsNewestTriggerLocalTimeMs * 1000);

    return GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

## 2. 注册获取 PPS 信号被触发时负载设备本地时间的函数

注册获取 PPS 信号被触发时负载设备本地时间功能的函数后，需要将该函数注册到负载设备控制程序中，当负载设备接收到 PPS 信号的上升沿时，基于 PSDK 开发的负载设备控制程序，能够记录 PPS 信号被触发时负载设备的本地时间。

```
// users must register getNewestPpsTriggerTime callback function
GduStat =
GDUTimeSync_RegGetNewestPpsTriggerTimeCallback(s_timeSyncHandler.GetNewestPpsTriggerLocalTime
Us);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("register GetNewestPpsTriggerLocalTimeUsCallback error.");
    return GduStat;
}
```

## 4. 时间同步

使用 PSDK 开发的负载设备控制程序通过 PsdkOsal\_GetTimeMs 获取负载设备上的本地时间并将负载设备上的时间转换为无人机系统的时间。

- 获取负载设备本地的时间

```
GduStat = osalHandler->GetTimeMs(&currentTimeMs);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("get current time error: 0x%08lX.", GduStat);
    continue;
}
```

- 时间转换

将负载设备本地的时间转换为无人机系统上的时间。

```
GduStat = GDUTimeSync_TransferToAircraftTime(currentTimeMs * 1000, &aircraftTime);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("transfer to aircraft time error: 0x%08lX.", GduStat);
    continue;
}
```

```
USER_LOG_DEBUG("current aircraft time is %d.%d.%d %d:%d %d %d.", aircraftTime.year,
aircraftTime.month, aircraftTime.day, aircraftTime.hour, aircraftTime.minute, aircraftTime.second,
aircraftTime.microsecond);
```



## 精准定位

**提示：** 在运行“精准定位”示例代码前，请使用 GduFlight2 或 MSDK 开发的 APP 查看无人机与 RTK 卫星间保持良好的通信状态，确保负载设备可获取精准的定位结果。

## 概述

为满足使用 PSDK 开发的负载设备对厘米级精度的定位需求，GDU 支持开发者使用基准站（如 D-RTK 2）和移动站（S400 RTK），借助 RTK（Real Time Kinematic，实时动态载波相位差分技术）获取无人机飞行姿态和高精度的定位信息。

## 基础概念

### 术语解释

- 目标点：实际获取到定位信息的位置，如云台口中心点。

**说明：** S400 RTK 的目标点为负载设备转接环上表面的中心点。

- 兴趣点：由用户任意指定的负载设备上某一器件的位置，如相机图像传感器的中心点，该目标点也可以为兴趣点。

- 任务：多个连续的飞行动作集合称为一个任务，如对某个区域执行一次测绘任务。根据实际使用需要，用户可创建多个任务。

- 定位事件：触发定位请求的事件，如相机曝光时触发定位请求，则“相机曝光”是一个定位事件；多个事件的集合为事件集合，使用 PSDK 开发的负载设备可同时请求多

个定位事件发生时的位置信息，如相机协同曝光。

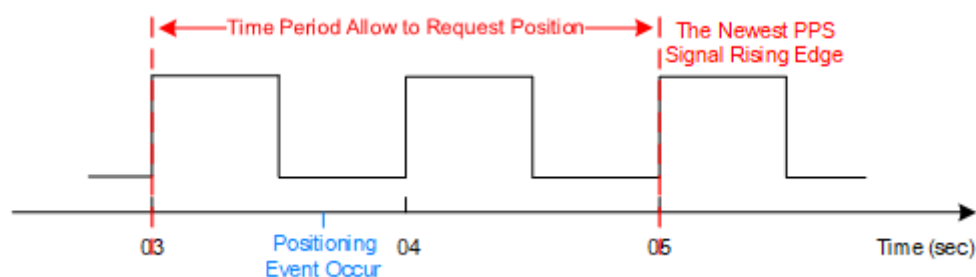
## 获取精准定位

**说明：** 获取精准定位时，需使用时间同步功能将负载设备的本地时间同步为无人机时间，有关使用时间同步功能的详细说明请参见时间同步章节。

1. 定位事件发生时，负载设备需要会记录本地时间（该时间为负载设备上的时间）；
2. 负载设备通过时间转换功能，将负载设备上的时间转换为无人机上的时间；
3. 负载设备使用定位事件发生时的无人机时间（无人机系统的时间）请求位置。

**说明:** 定位事件发生时的无人机时间（无人机系统的时间）应早于最新的 PPS 信号上升沿时间，且时间间隔须在 1 ~ 2s 内，如图 2.获取精准定位 所示。

图 2.获取精准定位



获取目标点的位置后，根据目标点的位置，使用目标点与无人机 RTK 主天线位置的偏移量、无人机的姿态、负载设备的结构等信息，能够计算兴趣点的位置。

## 使用精准定位功能

### 1. 定位功能模块初始化

使用“精准定位”功能前，需要先初始化精准定位功能模块，确保精准定位功能可正常运行。

```
GduStat = GduPositioning_Init();
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("positioning module init error.");
    return GduStat;
}
```

### 设置任务编号

**建议**使用任务编号功能，方便用户在 Mark 文件中快速查找到位置请求信息（未设置任务编号时，任务编号的默认值为 0）。

```
GduPositioning_SetTaskIndex(0);
```

### 2. 请求并打印位置信息

用户触发精准定位功能后，负载设备将根据定位事件发生时的时间获取精准的定位信息。

#### 1. 获取本地时间

用户触发精准定位功能后，负载设备获取 PPS 信号触发时负载设备上的本地时间。

```
GduStat = GduTest_TimeSyncGetNewestPpsTriggerLocalTimeUs(&ppsNewestTriggerTimeUs);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("get newest pps trigger time error: 0x%08lX.", GduStat);
    continue;
}
```

#### 1. 时间转换

将指定的定位事件发生时的负载设备上的本地时间转换为无人机系统的时间。

```

for (i = 0; i < GDU_TEST_POSITIONING_EVENT_COUNT; ++i) {
    eventInfo[i].eventSetIndex = s_eventIndex;
    eventInfo[i].targetPointIndex = i;

    GduStat = GDUTimeSync_TransferToAircraftTime(
        ppsNewestTriggerTimeUs - 1000000 - i * GDU_TEST_TIME_INTERVAL_AMONG_EVENTS_US,
        &aircraftTime);
    if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("transfer to aircraft time error: 0x%08lX.", GduStat);
        continue;
    }

    eventInfo[i].eventTime = aircraftTime;
}

```

## 2. 获取精准的定位信息

完成时间同步后，用户即可获取并打印负载设备在某一时刻的精准位置，负载设备详细的位置信息如 图 3. 定位详情 所示。

```

GduStat = GduPositioning_GetPositionInformationSync(GDU_TEST_POSITIONING_EVENT_COUNT,
    eventInfo, positionInfo);
if (GduStat != GDU_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("get position information error.");
    continue;
}

USER_LOG_DEBUG("request position of target points success.");
USER_LOG_DEBUG("detail position information:");
for (i = 0; i < GDU_TEST_POSITIONING_EVENT_COUNT; ++i) {
    USER_LOG_DEBUG("position solution property: %d.", positionInfo[i].positionSolutionProperty);
    USER_LOG_DEBUG("pitchAttitudeAngle: %d\trollAttitudeAngle: %d\tyawAttitudeAngle: %d",
        positionInfo[i].uavAttitude.pitch, positionInfo[i].uavAttitude.roll,
        positionInfo[i].uavAttitude.yaw);
    USER_LOG_DEBUG("northPositionOffset: %d\tearthPositionOffset: %d\tdownPositionOffset: %d",
        positionInfo[i].offsetBetweenMainAntennaAndTargetPoint.x,
        positionInfo[i].offsetBetweenMainAntennaAndTargetPoint.y,
        positionInfo[i].offsetBetweenMainAntennaAndTargetPoint.z);
    USER_LOG_DEBUG("longitude: %.8f\tlatitude: %.8f\theight: %.8f",
        positionInfo[i].targetPointPosition.longitude,
        positionInfo[i].targetPointPosition.latitude,
        positionInfo[i].targetPointPosition.height);
    USER_LOG_DEBUG(

```

```

        "longStandardDeviation: %.8f\\tlatStandardDeviation: %.8f\\thgtStandardDeviation: %.8f",
        positionInfo[i].targetPointPositionStandardDeviation.longitude,
        positionInfo[i].targetPointPositionStandardDeviation.latitude,
        positionInfo[i].targetPointPositionStandardDeviation.height);
    }

```

s\_eventIndex++;

图 3. 定位详情

```

[641.208][module_user]-[0.000][37m[Debug]- request position of target points success.[0.000][0m
[641.208][module_user]-[0.000][37m[Debug]- detail position information:[0.000][0m
[641.208][module_user]-[0.000][37m[Debug]- position solution property: 16.[0.000][0m
[641.208][module_user]-[0.000][37m[Debug]- pitchAttitudeAngle: 0 rollAttitudeAngle: 0 yawAttitudeAngle: 26[0.000][0m
[641.209][module_user]-[0.000][37m[Debug]- northPositionOffset: 241 earthPositionOffset: -153 downPositionOffset: 212[0.000][0m
[641.209][module_user]-[0.000][37m[Debug]- longitude: 113.935907 latitude: 22.525328 height: 167.598419[0.000][0m
[641.210][module_user]-[0.000][37m[Debug]- longStandardDeviation: 1.121413 latStandardDeviation: 0.968599 hgtStandardDeviation: 2.698592[0.000][0m
[641.210][module_user]-[0.000][37m[Debug]- position solution property: 16.[0.000][0m
[641.210][module_user]-[0.000][37m[Debug]- pitchAttitudeAngle: 0 rollAttitudeAngle: 0 yawAttitudeAngle: 26[0.000][0m
[641.211][module_user]-[0.000][37m[Debug]- northPositionOffset: 241 earthPositionOffset: -153 downPositionOffset: 212[0.000][0m
[641.211][module_user]-[0.000][37m[Debug]- longitude: 113.935907 latitude: 22.525328 height: 167.598428[0.000][0m
[641.212][module_user]-[0.000][37m[Debug]- longStandardDeviation: 1.121413 latStandardDeviation: 0.968599 hgtStandardDeviation: 2.698592[0.000][0m
[642.233][module_user]-[0.000][37m[Debug]- request position of target points success.[0.000][0m
[642.233][module_user]-[0.000][37m[Debug]- detail position information:[0.000][0m
[642.233][module_user]-[0.000][37m[Debug]- position solution property: 16.[0.000][0m
[642.233][module_user]-[0.000][37m[Debug]- pitchAttitudeAngle: 0 rollAttitudeAngle: 0 yawAttitudeAngle: 26[0.000][0m
[642.235][module_user]-[0.000][37m[Debug]- northPositionOffset: 241 earthPositionOffset: -153 downPositionOffset: 212[0.000][0m
[642.235][module_user]-[0.000][37m[Debug]- longitude: 113.935907 latitude: 22.525328 height: 167.595276[0.000][0m
[642.236][module_user]-[0.000][37m[Debug]- longStandardDeviation: 1.121065 latStandardDeviation: 0.968224 hgtStandardDeviation: 2.696033[0.000][0m
[642.236][module_user]-[0.000][37m[Debug]- position solution property: 16.[0.000][0m
[642.236][module_user]-[0.000][37m[Debug]- pitchAttitudeAngle: 0 rollAttitudeAngle: 0 yawAttitudeAngle: 26[0.000][0m
[642.236][module_user]-[0.000][37m[Debug]- northPositionOffset: 241 earthPositionOffset: -153 downPositionOffset: 212[0.000][0m
[642.237][module_user]-[0.000][37m[Debug]- longitude: 113.935907 latitude: 22.525328 height: 167.596619[0.000][0m
[642.238][module_user]-[0.000][37m[Debug]- longStandardDeviation: 1.121065 latStandardDeviation: 0.968224 hgtStandardDeviation: 2.696033[0.000][0m

```

## 适配产品

S400 RTK