# Kairos Software Development for Ardupilot

**20 Sept. 2024**

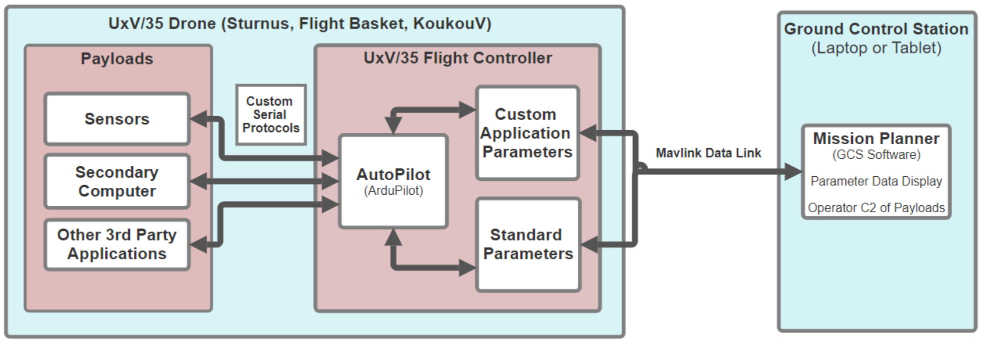
## Introduction

Ardupilot currently lacks flexibility in supporting custom unmanned aerial systems (UAS) payloads. This includes controlling payloads, transmitting data to ground control stations (GCS), and receiving commands from payloads. Kairos has therefore implemented the initial steps required for supporting these tasks. Documented in this paper are two modifications to the Ardupilot software.

The first task was to create new Mavlink parameters that can be modified on the UAS flight controller and transmitted to a GCS.

A second task was to implement a custom serial protocol that can be adapted to communicate with payloads on the UAS.

### Background on the Framework



Ardupilot is the autopilot software running on the UxV/35 Mission Controller (*KA1003-01*). While this software has a wide range of capabilities and features, Kairos and its customers are reaching a point that exceeds Ardupilots capabilities. The solution being implemented in this document is depicted in the diagram above.

By implementing custom serial protocols and parameters, Kairos provides a framework onboard the UxV/35 UAS to directly transmit and received data between payloads and GCSs. Payloads communicate with the flight controller through a custom serial protocol. The flight controller then processes this data to update custom parameters and optionally execute actions based on these updates. Payload data contained in the custom parameters is then transmitted to the GCS using the existing telemetry radio being used to control the drone. This payload data can be viewed on the GCS and commands can be sent back through the framework to control the payload.

Kairos believes that the Mavlink telemetry link is sufficient for many of these early tasks. Currently Kairos knows that the telemetry link can handle real-time concurrent transmission of 4 distance sensors while a UAS is flying. It should be acknowledged though that control of the drone is the first priority and that payload data transmission likely will reach a bandwidth limitation. Kairos does have plans to find this bandwidth limit through stress testing (long distances and large data transmission) and determine what the behavior is as the limit is approached. Safeguards will likely need to be established to prevent a payload from compromising the telemetry link.

## Creating New Parameters

Background:

Within the Ardupilot flight controller software are a long list of parameters used to configure and report the behavior of the UAS. Some of these parameters are static and used for initial setup, while others are updated at various frequencies to report status.

**Documentation of Changes:**

Adding new parameters requires changing of three files in the Ardupilot codebase.

Ardupilot/ArduCopter/Parameters.h

- Parameters are indexed under multiple classes. For testing, Kairos added a parameter to the ParametersG2 class.

```
AP_Int8 ka_custom;
```

Ardupilot/ArduCopter/Parameters.cpp

- The new parameter is then declared with descriptors used to set the range, and display information to a MissionPlanner GCS.

```
const AP_Param::Info Copter::var_info[] = {
   // @Param: KA_CUSTOM
   // @DisplayName: Custom KA Param
   // @Description: Used for custom KA Functions
   // @Range: 0 63
   // @User: Standard
   AP_GROUPINFO("KA_CUSTOM", 59, ParametersG2, ka_custom, 0),
}
```

Ardupilot/ArduCopter/config.h

- Finally, the default for the parameter is set

```
#ifndef KA_CUSTOM
        #define KA_CUSTOM           0 //Default
#endif
```

## Implementing New Serial Protocols

Background:

Ardupilot has a list of roughly 40 serial protocols for communicating with the flight controller. Each protocol is typically configured to communicate with a specific device or utility and execute a specific task on the flight controller.

**Documentation of Changes:**

Adding serial protocols is more involved, but revolves around the Ardupilot SerialManager. A library folder is then created for the new protocol and that folder is added to the list of dependencies compiled when Ardupilot is built.

<u>Ardupilot/libraries/AP_SerialManager (.cpp and .h)</u>

In SerialManager.h

- Define defaults for the new protocol

```
#define AP_SERIALMANAGER_KACUSTOM_BUFSIZE_RX       128
#define AP_SERIALMANAGER_KACUSTOM_BUFSIZE_TX       128
#define AP_SERIALMANAGER_KACUSTOM_BAUD      115200
```

- Add the new protocol to the list of options

```
SerialProtocol_KACustom = 45,
```

In SerialManager.cpp

- Add a case for initializing the protocol if selected

```
case SerialProtocol_KACustom:
   uart->begin(state[i].baudrate(),
               AP_SERIALMANAGER_KACUSTOM_BUFSIZE_RX,
               AP_SERIALMANAGER_KACUSTOM_BUFSIZE_TX);
   hal.console->printf("Initializing KACustom.\n");
   AP_KACustom::get_singleton()->init_serial(i);
   break;
```

<u>Ardupilot/libraries/AP_KACustom (.cpp and .h)</u>

In AP_KACustom.h

- Define public and private constructors, functions, variables, and UART driver

```
class AP_KACustom {
public:
   // Constructor
   AP_KACustom() {}

   // Initialize the serial protocol
   void init_serial(uint8_t serial_instance);
   // Update function that will be called regularly
   void update();
   static AP_KACustom* get_singleton();
protected:
   // Define the UART driver for serial communication
   AP_HAL::UARTDriver* uart = nullptr;
   // Baudrate initialization function
   virtual uint32_t initial_baudrate(uint8_t serial_instance) const;
   // Buffer size initialization
   virtual uint16_t rx_bufsize() const { return 128; }
   virtual uint16_t tx_bufsize() const { return 128; }
   // Helper function to send a test message
```

```
    void send_test_message();
    // This is a placeholder for future expansion when you want to get readings
or data
    virtual bool get_reading();
private:
    static AP_KACustom* _singleton;
};
```

In AP_KACustom.cpp

- Create functions to initialize an instance and send UART data

```
_KACustom* AP_KACustom::_singleton = nullptr;
AP_KACustom *AP_KACustom::get_singleton()
{
    if (!_singleton) {
        _singleton = new AP_KACustom();
    }
    return _singleton;
}
void AP_KACustom::init_serial(uint8_t serial_instance) {
    // Find the UART associated with the custom protocol
    uart =
AP::serialmanager().find_serial(AP_SerialManager::SerialProtocol_KACusto
m, 0);
    if (uart != nullptr) {
        //uart->begin(initial_baudrate(serial_instance), rx_bufsize(),
tx_bufsize());
    }
}
uint32_t AP_KACustom::initial_baudrate(uint8_t serial_instance) const {
    // Define the baudrate for the protocol
    return 115200;  // Example baud rate, customize as needed
}
void AP_KACustom::update() {
    if (uart != nullptr) {
        // Call the function to send a test message or process data
        hal.console->printf("AP_KACustom update() called.\n");
        send_test_message();
    }
}

void AP_KACustom::send_test_message() {
    if (uart != nullptr) {
        // Example of sending "test" message over the UART
        uart->printf("test\n");
    }
}
bool AP_KACustom::get_reading() {
    // Placeholder function for reading data, return false for now
    return false;
}
namespace AP {
    AP_KACustom& kac()
    {
        return *AP_KACustom::get_singleton();
    }
};
```

Ardupilot/Tools/ardupilotwaf.py

In ardupilotwaf.py

- Add AP_KACustom to the list of dependencies compiled by waf

```
COMMON_VEHICLE_DEPENDENT_LIBRARIES = [
        'AP_KACustom',
]
```

## Summary

While these first steps are a basic implementation, they should help demonstrate that this goal for a framework is very achievable. With further development, there is a powerful list of potential capabilities down the road.

- *Onboard transmission of payload data to GCS with no additional radio links*
- *Control of a payload from a GCS with no additional radio links*
- *Communication between payload and flight controller which allows for a payload controlled UAS and a UAS controlled payload*
- *Pathways to create new UAS behaviors*
- *Streamlined integration of new sensors, radios, and other technologies*
- *Standardized and documented interaction of payloads with Kairos drones to greatly simplify the development process for 3rd parties*

Kairos Autonomi
8700 S. Sandy Pkwy.
Sandy, Utah 84070
801-225-2950 (office)
801-907-7870 (fax)
www.kairosautonomi.com

## Version History

| Date and Signature | Revisions | Reasons for Revision |
|---|---|---|
| 09/20/2024 Jack R. | Document was written. (v01.00.00) | |
| 09/24/2024 Jack R. | Expansion of Background of Framework (v01.01.00) | Feedback on document |