# API documentation for PlutoX

Using Cygnus IDE


Explore. Experiment. Innovate.

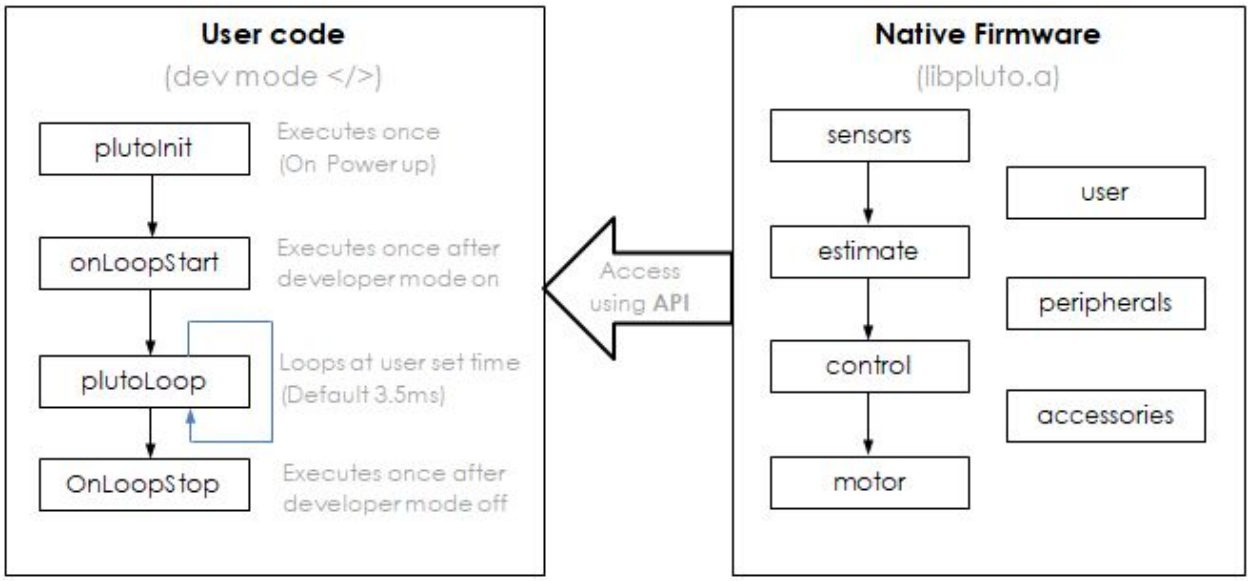
Version 1.1

# Introduction

## 1. Code Development

This document helps you program your PlutoX. The Native firmware is a fork Cleanflight firmware. Native firmware is responsible for basic drone functions. If you want to program the Native firmware you can refer to Ref F.

Using APIs access to critical flight variables and functions is provided. The structure of firmware is according to the following diagram. For a more details on firmware structure you can refer to Ref E. :




The User code execution starts when the drone is in developer mode. The developer mode can be controlled using the Pluto Controller app using the button **</>**. Please note that this button is accessible only when we choose "*I am a Developer*" from user profile in the App.
***Note**: The drone functions are already pre built in the pluto. In order to fly the drone, you don't need to code anything into the user code.*

There are four basic functions: <<

1. plutoInit()
   This function is executed immediately after the drone powers up. This function is particularly useful for initializing hardware.
2. onLoopStart()
   This function is executed once, after user the user turns the developer mode on. This function can be useful for initialising some variables

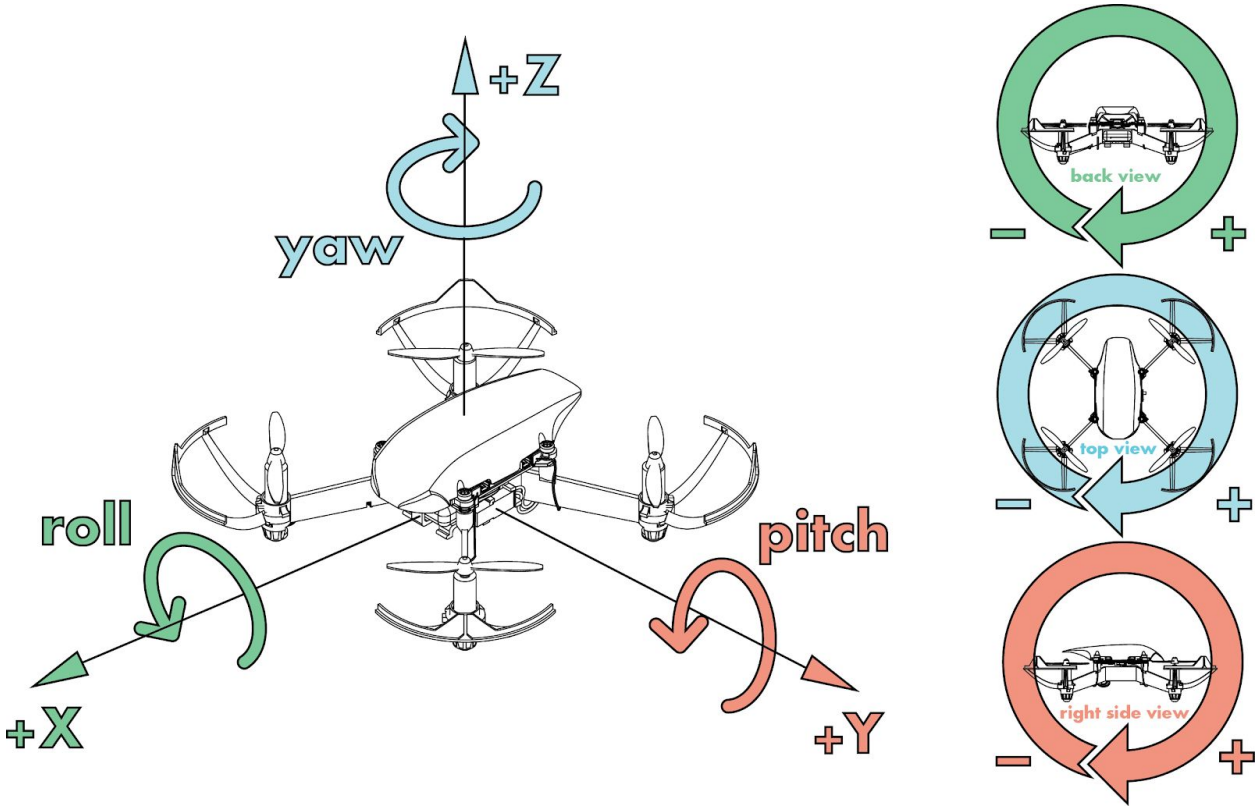
3. plutoLoop()
   This function is executed in loop, along with drones internal primary stabilization code. By default this loop runs every 3.5ms. Using APIs, you can modify this loop frequency. This will not affect the drones internal stabilization loop.
4. onLoopStop()
   This function is executed once, after the user turns off the developer mode.

While using the APIs it's important to note the frame of reference. In the following figure we briefly explain some basic angles and axis of the drone. If you need to know more about frame of references please refer to Ref C.

## 2. State variables

When you are trying to control a drone, there are many variables to choose from. These variables in state-space theory are referred as state variables. A State variable can be defined as the smallest set of variables that fully describe the system(drone) and its response to any given set of inputs. In case the system is a rigid body, there are 12 state variables that are used. These state-variables are:
**Position**: X, Y, Z
*These three variables represent position of the drone with respect to Inertial frame.*
**Velocity**: Vx, Vy, Vz
*These three variables represent velocity of the drone with respect to Inertial frame.*
***Rotation rate****: p, q, r*
*These three variables represent the rotational rates of the drone with respect to body-axis.*
**Angles**: Roll($\varphi$), Pitch($\theta$), Yaw($\psi$)
*There three variables represent the orientation of the drone with respect to inertial frame. These three angles are non-commutative. Thus the sequence of rotation matters. This representation is called 3-2-1 Euler representation.*
*As seen in the figure below:*

- *Roll represents rotation about X-axis*
- *Pitch represents rotation about Y-axis*
- *Yaw represents rotation about Z-axis*




## 3. General guidelines

- The function plutoPilot is executed in one of the main control loops of the firmware. Thus **don't write blocking code**.
- To use delay, use the timer functions. The standard delay function are blocking in nature.
- Please note that most of the classes are predefined. You do not need to define the class to use it unless mentioned in the Documentation.

# Sensors:

## 1. Accelerometer

***Description***: An accelerometer doesn't actually provide acceleration. It rather provides “proper acceleration”.
**Scope:** plutoOnLoopStart(), plutoLoop(), plutoOnLoopFinish()

i. `int16_t get(axis_e AXIS)`
*Description*: Gets X / Y / Z axis component of “proper acceleration”.
***AXIS***: *X, Y, Z*
***Units***: cm/$s^2$

ii. `int32_t getNetAcc(void)`
*Description*: This is the RMS value of the net acceleration. It is defined by sqrt($a_x^2 + a_y^2 + a_z^2$).
***Units***: cm/$s^2$

*Example*: To access the net magnitude of acceleration of the drone, use:

```
int32_t rmsAcc, Ax;
rmsAcc = Acceleration.getNetAcc();
Ax = Acceleration.get(X);
```

## 2. Gyroscope

***Description***: The gyroscope on the drone is a rate gyro. Thus it doesn’t provide angles, but rather the instantaneous angular rates about body axis.

i. `int16_t get(axis_e AXIS)`
*Description*: Gets X / Y / Z axis component of gyro.
***AXIS***: *X, Y, Z*
***Units***: deciDegree/s. 10 decidegree = 1 degree.

***Example***: To access the angular rate about drones X-axis, use:

```
int16_t sample;
sample = Gyroscope.get(X);
```

## 3. Magnetometer

***Description***: In order to get an accurate sense of yaw & heading of the drone has a magnetometer. The yaw and heading of the drone is extremely critical during navigation.

***Note***: In order to get relevant data, it extremely important to calibrate the magnetometer. Please refer to the user manual for the same.

I. `int16_t get(axis_e AXIS)`
*Description*: Gets X / Y / Z axis component of mag.
***AXIS****: X, Y, Z*
***Units***: microTesla.

***Example***: To access the magnetic field about drones X-axis, use:

```
int16_t sample;
sample = Magnetometer.get(X);
```

## 4. Barometer

***Description***: The onboard barometer measures pressure along with temperature sensor. This is very critical to estimate the drones height.

I. `int32_t get(baro_state_e STATE)`
*Description*: Gets absolute pressure/temperature.
***STATE****: PRESSURE, TEMPERATURE*
***Units****(pressure)*: 100*millibar.
For example if your reading is 100009. This is equal to 1000.09 mBar

***Units****(temperature)*: 100*degree celsius.
For example if your reading is 2009. This is equal to 20.09 degrees celsius.

***Example***: To access current absolute temperature and pressure:

```
int32_t pressure, temp;
pressure = Barometer.get(PRESSURE);
temp = Barometer.get(TEMPERATURE);
```

# Estimate:

Most of the sensor data is unusable without filtering. In pluto, a complementary filter is used to estimate various states of the drone.

## 1. Angle

***Description***: Returns Euler current angle. The representation is in 3-2-1 format.
***Frame:*** Yaw - Inertial, Pitch - Intermediate 1, Roll - Intermediate 2.

```
int16_t get(angle_e ANGLE)
```

***ANGLE***: *AG_ROLL, AG_PITCH, AG_YAW*
***Units***:
Roll represented in deciDegree. Ranges : -1800 to 1800 deciDegrees
Pitch represented in deciDegree. Ranges : -900 to 900 deciDegrees
Yaw represented in Degree. Ranges : 0 to 360 Degrees
10 deciDegree = 1 degree

***Example***: To access the drones roll, you can use:

```
int16_t roll;
roll = Angle.get(AG_ROLL);
```

## 2. Rate

***Description***: This represents filtered body axis rotation rates(p, q, r).
***Frame***: Body frame.

```
int16_t get(axis_e AXIS)
```

***AXIS***: *X, Y, Z*
***Units***: deciDegree/s.

***Examples***:

```
int16_t p, q, r;
p = Rate.get(X); //This represents p
q = Rate.get(Y); //This represents q
r = Rate.get(Z); //This represents r
```

***Note***: The angles and rotation rates are in different frames of reference. Thus:
$\frac{d}{dt}\phi \neq p;\ but\ \frac{d}{dt}\phi \simeq p$
$\frac{d}{dt}\theta \neq q;\ but\ \frac{d}{dt}\theta \simeq q$

$\frac{d}{dt}\varphi \neq r;\ but\ \frac{d}{dt}\varphi \simeq r$
This approximation is valid for small angles in roll and pitch axis.

## 3. Position

***Description***: This represents accesses the current position of drone.
***Frame***: Inertial frame

```
int16_t get(axis_e AXIS)
```
***AXIS***: X[1], Y[1], Z
***Units***: cm

***Examples***:

```
int16_t Height;
Height = Position.get(Z); //This shows absolute height
```

***Note***:

1. The Height of the drone is expressed in cm. The reference "zero" is the height at which the drone is turned on.
2. The accuracy of Height estimate depends on the accuracy of the sensor. In case only onboard barometer is used, you can expect an accuracy of +/- 50cm
3. The **X and Y position of the drone will not return valid value, unless external referencing sensors are used**. Example of external reference are: GPS, UWB, opticflow, WhyCon etc.

## 4. Velocity

***Description***: This represents filtered body axis rotation rates(p, q, r).
***Frame***: Inertial frame

```
int16_t get(axis_e AXIS)
```
***AXIS***: X[1], Y[1], Z
***Units***: cm/s

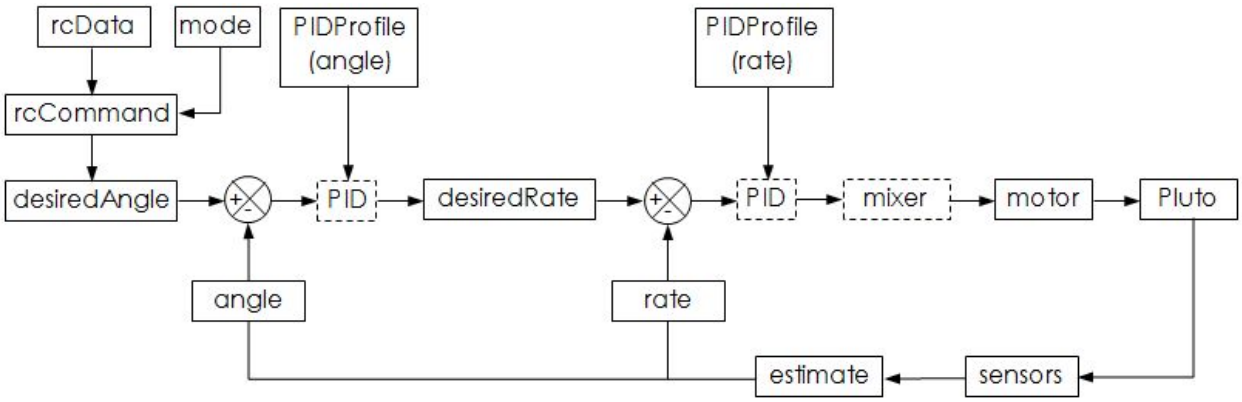
***Examples***:

```
int16_t verticalVelocity;
verticalVelocity = Velocity.get(Z);
```

[1] Please use this should be accessed only when additional localization sensors(GPS, opticflow, UWB, etc) are used.

# Control:

This section is meant for access to various control variables. The control loop structure changes slightly depending on flight modes. An example of **angle mode** control is provided below.




**Note:** As seen in the figure above, in Angle model the control variables are dependent as follows:

rcCommand -> desiredAngle -> desiredRate -> motors

The symbol "->" refers to "generates". Thus, if "rcCommand" is set using the APIs, manipulating "motors" doesn't make sense. Hence it is recommended that you don't command different control variables at the same time.
If multiple control variables from different groups are used, the firmware chooses the control variable with the highest priority. The priority of control variable is as follows(descending order of priority):

1. motor
2. desiredRate
3. desiredAngle
4. desiredVelocity
5. desiredPosition
6. rcCommand

## 1. DesiredAngle

***Description***: This API gives access to desired angle (setpoint).
***Frame:*** Yaw - Inertial, Pitch - Intermediate 1, Roll - intermediate 2.

i. `int32_t get(angle_e ANGLE)`
*Description*: Get the angle setpoint.
***ANGLE****: AG_ROLL, AG_PITCH, AG_YAW*

ii. `void set(angle_e ANGLE, int32_t angle)`
*Description*: Set the angle setpoint
***ANGLE****: AG_ROLL, AG_PITCH, AG_YAW*
***Units***: deciDegrees

***Examples***:

```
int32_t rollSetpoint;
rollSetpoint = DesiredAngle.get(AG_ROLL);
```

## 2. DesiredRate

***Description***: This API gives access to rate setpoint.
**Frame**: Body frame

I. `int32_t get(angle_e ANGLE)`
*Description*: Get the rate setpoint.
***ANGLE****: AG_ROLL, AG_PITCH, AG_YAW*

ii. `void set(angle_e ANGLE, int32_t angle)`
*Description*: Set the rate setpoint.
***ANGLE****: AG_ROLL, AG_PITCH, AG_YAW*
***Units***: deciDegree/s

***Examples***:

```
int32_t rollRateSetpoint;
rollRateSetpoint = DesiredRate.get(AG_ROLL);
```

## 3. DesiredPosition

***Description***: This API gives access to rate setpoint.
**Frame**: Inertial frame

i. `int32_t get(axis_e AXIS)`

*Description*: Get the position setpoint.
***AXIS***: *X[2], Y[2], Z*

ii. `bool set(axis_e AXIS, int32_t position)`
*Description*: Set the position setpoint. It returns true, when the control variable achieves desired position.
***AXIS***: *X[2], Y[2], Z*

iii. `bool setRelative(axis_e AXIS, int32_t position)`
*Description*: Set the relative position setpoint. It returns true, when the control variable achieves desired position.
***AXIS***: *X[2], Y[2], Z*
***Units***: cm

**Example**:

```
int32_t height;
height = DesiredPosition.get(Z);
//Alternatively, this can also be used
height = DesiredPosition.set(Z);
//This API is useful to change the relative, height to 100 cm higher than
//current height
desiredPosition.set(Z, 100);
```

## 4. DesiredVelocity

***Description***: This API gives access to velocity setpoint.
**Frame**: Body frame

i. `int32_t get(axis_e AXIS)`
*Description*: Get the velocity setpoint.
***AXIS***: *X[2], Y[2], Z*

ii. `bool set(axis_e AXIS, int32_t velocity)`
*Description*: Set the velocity setpoint.
***AXIS***: *X[2], Y[2], Z*
***Units***: cm/s

***Examples***:

```
int32_t height;
rollRateSetpoint = DesiredVelocity.get(Z);
```

[2] Please use this should be accessed only when additional localization sensors(GPS, opticflow, UWB, etc) are used.

## 5. PIDProfile

i. `void get(pid_profile_e PROFILE, PID* pid)`
*Description*: Get the PID gain values of the state variable.
***STATE**: PID_ROLL, PID_PITCH, PID_YAW, PID_ALT, PID_USER*

ii. `void set(pid_profile_e PROFILE, PID* pid)`
*Description*: Set the PID gain values for angle.
***STATE**: PID_ROLL, PID_PITCH, PID_YAW, PID_ALT, PID_USER*

iii. `void setDefault()`
*Description*: Set the default PID gain values for all states.

***Examples***:

```
pid anglePID; //This is a structure of pid variable.
uint8_t P, I, D;
PIDProfile.get(ROLL, &anglePID);
//You can access the individual gains as follows
P = anglePID.p;
I = anglePID.i;
D = anglePID.d;
```

## 6. Failsafe

i. `void enable(failsafe_e FAILSAFE)`
*Description*: This function is used to enable particular failsafe state.
***STATE**: LOW_BATTERY, INFLIGHT_LOW_BATTERY, CRASH, ALL*

ii. `void disable(failsafe_e FAILSAFE)`
*Description*: This function is used to disable particular failsafe state.
***STATE**: LOW_BATTERY, INFLIGHT_LOW_BATTERY, CRASH, ALL*

***Examples***:

```
Failsafe.disable(CRASH);/This API is useful for controlling failsafe

Failsafe.enable(ALL);
```

# User:

## 1. RcData

***Description***: This API provides access to raw unfiltered Remote control data. Pluto by default supports 8 channels.
***Range***: 900 to 2100
Low: 1000
Mid: 1500
High: 2000
***Channel allocation***:

| | | |
|---|---|---|
| 1: Roll | 5. AUX1 | 9. USER1 |
| 2: Pitch | 6. AUX2 | 10.USER2 |
| 3: Yaw | 7. AUX3 | 11.USER3 |
| 4: Throttle | 8. AUX4 | |

i. `int16_t* get()`
*Description*: Returns the current value of the 11 channel RCData array.

ii. `int16_t get(rc_channel_e CHANNEL)`
*Description*: Returns the current value of the RCData of particular chanel .
***CHANNEL****: RC_ROLL, RC_PITCH, RC_YAW, RC_THROTTLE, RC_AUX1, RC_AUX2, RC_AUX3, RC_AUX4, RC_USER1, RC_USER2, RC_USER3*

***Example:***

```
int16_t rc[11];
int16_t rc_roll_stick, rc_throttle_stick;
RcData.get();
rc_roll_stick = rc[0]; //This gives you the roll stick position
rc_throttle_stick = rc[3]; //This gives you the throttle stick position
```

## 2. RcCommand

**Description**: This API provides access to Remote control command. The RCCommand is a modified rcData after applying filters like low-pass filter, expo etc. It also changes the range to user friendly range.
Note: rcCommand is applicable to only first four channels.
***Range***: -500 to 500
Low: -500
Mid: 0
High: 500
***Channel allocation***:

1: Roll
2: Pitch
3: Yaw
4: Throttle

i. `int16_t* get()`
*Description*: Returns the current value of the 4 channel RcCommand array.

ii. `int16_t get(rc_channel_e CHANNEL)`
*Description*: Returns the current value of the RcCommand of particular chanel.
***CHANNEL**: RC_ROLL, RC_PITCH, RC_YAW, RC_THROTTLE*

iii. `void set(int16_t* rcValueArray)`
*Description*: Sets the value of the 4 channel RCCommand array.

iv. `void set(rc_channel_e CHANNEL, int16_t rcValue)`
*Description*: Sets the value of particular channel of the RCCommand array.
***CHANNEL**: RC_ROLL, RC_PITCH, RC_YAW, RC_THROTTLE*

***Example:***
To manipulate the roll stick, use the following code.

```
int16_t rcCmd = 1000;
RcCommand.set(RC_ROLL, rcCmd);
```

## 3. FlightMode

***Description***: This API provides access to different flight modes of Pluto.
***Available modes***:
*Group 1*. Angle mode, Rate mode
*Group 2*. Headfree
*Group 3*. Throttle, Altitude hold
***Note***:

- In every group, you can choose only one mode at a time. For example you can't set the drone in rate mode and angle mode at the same time.
- The default mode is Angle mode & Altitude hold mode.
- Various flight modes are explained in Ref A.

i. `bool check(flight_mode_e MODE)`
*Description*: Checks if the particular flight mode is active.
***MODE**: ANGLE, RATE, MAGHOLD, HEADFREE, ALTITUDEHOLD, THROTTLE_MODE*

ii. `bool set(flightstatus_e  status)`
*Description*: Set the flight mode form given groups.
***MODE**: ANGLE, RATE, MAGHOLD, HEADFREE, ALTITUDEHOLD, THROTTLE_MODE*

***Example:***
This code makes the drone fly in rate mode & throttle mode. Such a mode is generally used by drone racing pilots.

```
FlightMode.set(RATE);
FlightMode.set(THROTTLE);
```

## 4. Command

***Description***: These APIs gives access to basic drone commands

i. `void takeOff(uint16_t height=150)`
*Description*: This function is used to take-off at a default height of 150cm. It returns 0, when the takeoff is complete.
***Units***: cm
***Height***: *100 to 250*

iii. `void land(uint8_t landSpeed=105)`
*Description*: This function is used to land pluto at default speed of 105. It returns 0, when the land is complete.
***Speed***: *0 to 255*

iii. `void flip(flip_direction_e direction)`
*Description*: This function is used to flip the drone.
***Direction***: *BACK_FLIP*

iv. `bool arm()`
*Description*: This function is used to arm the drone. Arming essentially controls power to the motors.
*Note*:

- This function requires the ARM switch on the APP/RC to be turned on.
- If the ARM switch is turned off, this function will not work.
- This dependency is useful to ensure can be switched off from APP / RC in case of emergency.

v. `bool disArm()`
*Description*: This function is used to DisArm the drone. Arming essentially controls power to the motors.

***Example***:

```
Command.takeOff(200);
Command.land(105);
```

```
Command.flip(BACK_FLIP);
```

## 5. User Loop

***Description*:** This API lets you configure the speed of plutoLoop() function. By default this loop runs every 100 millisecond.

i. `void setUserLoopFrequency(float frequency)`
**interval :** *3.5 - 2000.*
The units of this interval are in ms.

***Example***:
To write a piece of code to blink the red led once a second:

```
void plutoInit(){
   setUserLoopFrequency(1000);
}

void onLoopStart(){
   LED.flightStatus(DEACTIVATE);
}

void plutoLoop(){
   LED.set(RED, TOGGLE);
}
void onLoopFinish(){
   LED.flightStatus(ACTIVATE);
}
```

## 6. App

***Description:*** This section of APIs help you interact with Pluto's App.

i. `int16_t getAppHeading(void)`
*Description*: This function gives you current heading of your phone
***Units***: Degree.
***Ranges*** *: 0 to 360 Degrees.*

ii. bool isArmSwitchOn(void)
*Description*: This function checks the status of arm switch. It returns true when the switch is on.

## Utils

This section of APIs are useful for debugging the code.

### 1. LED

***Description***: This API gives access to the onboard LEDS

i. `void set(led_e LED, led_state_e STATE)`
*Description*: This API allows you to set a particular Led or combination of Leds
***LED***: *RED, BLUE, GREEN*
***STATUS***: *OFF, ON, TOGGLE*

ii. `void flightStatus(flightstatus_state_e STATE)`
*Description*: This API allows you to control the Flight status of the drone
***STATUS***: *ACTIVATE, DEACTIVATE*

***Note***: By default, the LEDs are used to show the drone status. It is recommended that you use the "`FlightStatus(DEACTIVATE)`" command to disable the LEDs being used for the drones status.

***Example***:

```
LED.flightStatus(DEACTIVATE); //Makes sure that leds not showing flight status.
LED.set(RED, ON);
LED.set(BLUE, TOGGLE);
LED.set(GREEN, ON);
```

### 2. Monitor (Print)

***Description***: This function is very useful do debug you code. In general you should not use this function directly. This function works best if printed at intervals above 100ms.

i. `void print(const char* msg);`
*Description*: This function is used to print a message on Pluto Monitor.

ii. `void print(const char* tag, int number)`
*Description*:This function is used to print integer number on Pluto Monitor

iii.`void print(const char* tag, double number, uint8_t precision)`

*Description*:This function is used to print float number on Pluto Monitor

iv.`void println(const char* tag)`
*Description*:This function is used to print a message on a new line on Pluto Monitor

v.`void println(const char* tag, int number)`
*Description*:This function is used to print integer number on a new line on Pluto Monitor

iv.`void println(const char* tag, int number, uint8_t precision)`
*Description*:This function is used to print float number on a new line on Pluto Monitor

***Example***: To print message every 1000ms:

```
Interval T1;

void plutoLoop(){
   bool flag;
   int16_t roll_angle;

   roll_angle = Angle.get(AG_ROLL);
   flag = T1.set(1000,TRUE)
   if(flag){
      Monitor.print("Hi there... plutoX here..." );
      Monitor.println("This is how I roll ", roll_angle);
   }
}
```

## 3. Graph

***Description***: This function is used to monitor any data using graphical method.

i. `void red(double value, uint8_t precision=4)`

ii. `void green(double value, uint8_t precision=4)`

iii. `void blue(double value, uint8_t precision=4)`

*Description*:This function is used to print data on graphs. At a time 3 graphs can be observed

***Example***:

```
float roll_angle = Angle.get(AG_ROLL));
Graph.red(roll_angle, 0); //
```

## 4. Interval

***Description***: These API are very useful for executing periodic tasks at a particular interval.
***Note:*** Please note that you have to  declare interval object.

i. `bool set(uint32_t time, bool repeat)`
*Description*: This is used to set the interval that the user desires.

ii. `void reset(void)`
*Description*: This is used to reset the timer of your instance.

***Example:***

```
#include "PlutoPilot.h"
#include "utils.h"

Interval T1;

void plutoInit()
{
}

void onLoopStart()
{
   T1.reset();//This is not necessary. If not declared, the interval's
timer keeps running in the background.
}

void plutoLoop()
{
   bool flag;
   flag = T1.set(1000,TRUE)
   if(flag){
      print("Hi");
   }
}
```

## 5. System time

***Description***: This is useful for accessing the system's time in microseconds.

i. `uint32_t micros()`
*Description*: This returns the current system time.
The system time is returned in microseconds.

ii. `uint32_t millis()`
*Description:* The system time is returned in milliseconds.

***Example:***

```
uint32_t system_time;
system_time = micros();
```

# Motor

***Description***: These set of APIs are used to control motors M1-M8. There are two sets of motor drives available with PrimusX. Motors M1 - M4 are H-bridge drives(reversible) and thus can rotate motors in clockwise as well as counter-clockwise. Motors M5 - M8 are regular MOSFET drives.

i. `void init(motor_e motor)`
*Description*: This function initialises timers of the motor
***MOTOR***: *M1, M2, M3, M4, M5, M6, M7, M8*

ii. `void initReversibleMotors()`
*Description*: If this function is enabled this uses motors M1-M4 to flying in quad-X configuration. These motors have an advantage that they can be reversed inflight for inverted flight.

iii. `void set(motor_e motor, int16_t pwmValue)`
*Description*: This function can be used to control the power to the motors.
***MOTOR***: *M1, M2, M3, M4, M5, M6, M7, M8*
***Value***: *1000 - 2000*

iv. `void setDirection(motor_e motor, motor_aerial_direction_e direction)`
*Description*: This API is for reversing the direction of rotation of motor drive. It can only be used for M1-M4.
This set is useful, when you are using the drives to fly Pluto.
The motors will rotate clockwise / counter clockwise as seen from the top.
***MOTOR***: *M1, M2, M3, M4*
***DIRECTION***: *CLOCK_WISE, COUNTERCLOCK_WISE*

v. void setDirection(motor_e motor, motor_terrestrial_direction_e direction)
*Description*: This API is for reversing the direction of rotation of motor drive. It can only be used for M1-M4. This is set is useful, when you are using the drive to run motors on ground.
***MOTOR***: *M1, M2, M3, M4*
***DIRECTION***: *FORWARD, REVERSE*

***Example:***

```
Motor.init(M1);
Motor.init(M2);
Motor.setDirection(M1, FORWARD);
Motor.setDirection(M2, FORWARD);
Motor.set(M1, 1000);
Motor.set(M2, 1000);
```

# Peripherals:

These set of APIs are useful for controlling peripherals of the microcontroller using the UniBus. Each pin is assigned a reference number for ease of access. One of the recommended ways of accessing the pins is using the X-Breakout accessory board. Please refer to the figure below %%

## 1. GPIO

**Description**: This API is useful when user wants to use the UniBus pin as digital input or output pins.

i. `void init(unibus_e pin_number, GPIO_Mode_e mode)`
*Description*: This function is used to access the unibus pins and assign it any the modes available.
Pin: 2, 3, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19
Mode: INPUT, INPUT_PU, INPUT_PD, OUTPUT

ii. `bool read(unibus_e pin_number)`
*Description*: This function is used to read data from a particular pin of unibus.
**PIN:** 2, 3, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19

iii. `void write(unibus_e pin_number,GPIO_State_e STATE)`
*Description*: This function is used to write data to a particular pin of unibus.
**STATUS**: STATE_LOW, STATE_HIGH, STATE_TOGGLE

***Example***:

```
void plutoInit(){
   GPIO.init(10, OUTPUT);
   setUserLoopFrequency(1000);//set the loop interval to 1 second
}

void plutoLoop(){
   GPIO.output(10, TOGGLE);
}
```

## 2. ADC

**Description:** This API is useful while integrating an external analog device.

i. `void init(unibus_e pin_number)`

*Description*: This function is used to initialize an ADC pin
**PIN:** 8, 13, 14, 15, 16, 17, 18, 19

ii. uint16_t read(unibus_e pin_number)
*Description*: This function is used to read digital data from ADC pins
**PIN:** 8, 13, 14, 15, 16, 17, 18, 19

***Example***:

```
void plutoInit(){
   ADC.init(13);
}

void plutoLoop(){
   ADC.read(13);
}
```

## 3. PWM

**Description:** This API is useful while integrating external device requiring PWM.
Eg. Motors, Servos, etc.

i. `void init(unibus_e pin_number, uint16_t pwmRate)`
*Description*: This function is used to initialize a unibus pin and assign pwmRate.
**PIN:** 2, 3, 4, 5, 8, 9, 10, 13, *15, 18, 19*
**Freq**: 50 - 20000 ( Units: Hz)

ii. `void write(unibus_e pin_number, uint16_t pwmValue)`
*Description*: This function is used to set PwmValue to a unibus pin
**PIN:** 2, 3, 4, 5, 8, 9, 10, 13, *15, 18, 19*
**Value**: 500 - 2500 ( Units: Hz)

***Example***:

```
void plutoInit(){
   PWM.init(2, 50);
}

void plutoLoop{
   PWM.write(2, 1000);
}
```

## ***4. UART

**Description:** This API is useful while integrating devices with UART communication.

i. `void init(UART_Baud_Rate_e BAUD)`
*Description*:
**BAUD:** 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000 and 256000

ii. uint8_t read8()
*Description*:

iii. uint16_t read16()
*Description*:

iv. uint32_t read32()
*Description*:

v. void write(UART_Port_e PORT, uint8_t data)
*Description*:

vi. void write(UART_Port_e PORT, uint8_t* data, uint16_t length)
*Description*:

vii. bool rxBytesWaiting(UART_Port_e PORT)
*Description*:

viii. bool txBytesFree(UART_Port_e PORT)
*Description*:

## ***5. I2C

**Description:** This API is useful while integrating devices with I2C communication.

i. `uint8_t* read(uint8_t device_add, uint8_t reg, uint32_t length)`
*Description*:

ii. `bool write(uint8_t device_add, uint8_t reg, uint32_t length, uint8_t* data)`
*Description*:

# Accessories:

## 1. X-Ranging

**Description**: This API is useful for accessing X-Ranging Shield.
Note: Always ensure that, all the sensor that are initialized are also present on the X-Ranging shield

i. `void init()`
Description: Initializes all the 4 sensors.

ii. `void init(laser_e laser)`
Description: This function is used to initialize particular sensor.
***LASER**: LEFT, RIGHT, FRONT, BACK*

iii. `int16_t getRange(laser_e laser)`
Description: This function is used to get the instantaneous range from particular sensor.
***LASER**: LEFT, RIGHT, FRONT, BACK*

***UNIT:** mm.*

# Reference

## Ref A: Flight modes <<

Group 1:

Angle mode:
Rate mode:

Group 2:

Headfree mode

Group 3:

Throttle mode:
Altitude hold mode:

## Ref D: Unibus <<

Unibus (20 Pins)

- Power pins - 4
  - ➢ Pin 1 - VBAT
  - ➢ Pin 11 - GND
  - ➢ Pin 12 - 5 V
  - ➢ Pin 20 - 3.3 V

- GPIO - 16
  Pin 2, 3, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19.

- Analog Channel - 8
  Pin 8,13,14,15,16,17,18,19.

- DAC - 2
  Pin 8,13.

- UART
  - ➢ Pin 18 - RX
  - ➢ Pin 19 - TX

- I2C
  - ➢ Pin 7 - SDA
  - ➢ Pin 8 - SCL

- SPI
  - ➢ Pin 14 - SS
  - ➢ Pin 15 - SCK
  - ➢ Pin 16 - MISO
  - ➢ Pin 17 - MOSI

- Timer
  Pin 2,3,4,5,8,9,10,13,15,16,17,18,19

“Rub two sticks together long enough.... Fast enough...
your palms will blister... you will rub the skin right off.. Your jaw will clench from the pain...
you will fail and fail again... but then...
Where there was darkness... there is light...
Where there was cold... there is warmth...
You can see every corner of the cave...
You can walk into the night... even without the moon or stars to guide you....
But to get there you had to believe...
You had to believe that there was fire in the dead wood that made your fingers bleed...
No it wasn't god that lit the way... it was your own two hands...”

Make. Believe.

DRONA™
AVIATION