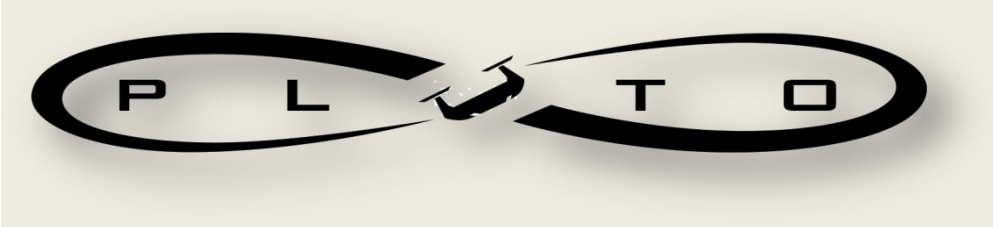

Pluto-X API (V.0.1) Documentation

1: Control APIs-

(i) bool isOkToArm(void)

- **Return's true when Pluto has finished its initial calibrations and now ready to arm i.e. to start motors.**

(ii) bool isArmed(void)

- **Return's true when Pluto has been armed.**

(iii) bool arm(void)

- **Start's the motors and return's true on success.**

(iv) bool disArm(void)

- **Stop's all the motors and return's true on success.**

(v) void setRC(rc_channels_e channel, int16_t value)

- **Set RC channel value.**
- **Available channels with min and max value respectively as follows:**
  1: RC_THROTTLE (1000, 2000) – in Althold mode 1500 is neutral value, below 1500 to 1000 Pluto will come downwards and above 1500 to 2000 Pluto will go upwards, in Throttle mode Pluto will go upwards at any throttle value.

  2: RC_ROLL (1000, 2000) - In both Althold and Throttle mode 1500 is neutral value, from 1500 to 1000 Pluto will roll to left and from 1500 to 2000 Pluto will roll to right.

3: RC_PITCH (1000,2000)- In both Althold and Throttle mode 1500 is neutral value, from 1500 to 1000 Pluto will pitch to backward and from 1500 to 2000 Pluto will pitch to forward.

4: RC_YAW (1000, 2000) - In both Althold and Throttle mode 1500 is neutral value, from 1500 to 1000 Pluto will yaw to left and from 1500 to 2000 Pluto will yaw to right.

Note: Pluto has two flying modes

1. **Althold mode- in which Pluto maintains the height where you leave the throttle control i.e set to 1500**
2. **Throttle mode- in doesn't maintain any height when you leave throttle, you need to continuesly provide throttle input in keep it upward.**

(vi) int16_t getRC(rc_channels_e channel)

-Returns the current value RC channel

(vii) void disableFlightStatus(bool disable)

- **Disable's default LEDs functions used for indication of different flight status if set true.**

(viii) bool checkFlightStatus(f_status_e status)

- **Check if status is currently active or not.**
- **Available flight status are as follow:**

  1: FS_Accel_Calibration- becomes active when Pluto requires accelerometer calibration

  2: FS_Mag_Calibration- becomes active when Pluto requires magenetometer callibration

  3: FS_Low_battery- becomes active when Pluto has battery below 3.4 V

  4: FS_LowBattery_inFlight- becomes active when Pluto reaches battery below 3.1 V while flying

  5: FS_Crash- becomes active when Pluto crashes.

  6: FS_Signal_loss- becomes active when there is communication loss between Pluto and mobile App.

  7: FS_Not_ok_to_arm- becomes active when Pluto is not ready to arm (because of things like it needs calibration or need to keep on plane surface).

  8: FS_Ok_to_arm- becomes active when Pluto is ready to arm.

  9: FS_Armed- becomes active when Pluto is armed.

(ix) f_status_e getFlightStatus(void)

- **Get current active flight status.**

(x) void setFailsafe(failsafe_e failsafe, bool activate)

- **Enable or disable different failsafe's (LOW_BATTERY, INFLIGHT_LOW_BATTERY, CRASH, ALL).**
- **All failsafe's are enabled by default.**

(xi) void setUserLoopFrequency(uint16_t frequency)

- **Set's plutoPilot function frequency in milliseconds**
- **Allowed range is 0ms-300ms: default is 100ms**

Usage: To access Control APIs you have to use predefined object 'Control'.

Examples: 1- Control.setRC(RC_THROTTLE, 1700) ; setting throttle to 1700

2- Control.setFailsafe(CRASH, false); disabling the crash failsafe check

3- Control.arm(); armes the Pluto;

2: AltHold APIs-

(i) int32_t getEstimatedAltitude(void)

- **Return's height (cm) estimated for althold.**

(ii) int32_t getVelocityZ(void)

- **Return's Z-axis velocity (cm/sec) in used by althold.**

(iii) void activateAlthold(bool activate)

- **Activate althold if set true: if set false disable althold and starts throttle mode.**

  Note: althold is default mode.

(iv) bool isAltholdModeActivate(void)

- **Return's true if Pluto is in althold mode else false on throttle mode.**

(v) void setAltholdHeight (int32_t height)

- **Set's height (cm) for althold.**

(vi) void setRelativeAltholdHeight (int32_t height)

- **Set's height (cm) for althold relative to Pluto's current height.**

(vii) Int32_t getAltholdHeight (void)

- **Get current althold height.**

Usage: To access Althold APIs you have to use predefined object 'Alhold'.

Examples: 1- Althold.activateAlthold(); //activates althold mode

2- Althold. getVelocityZ(); // gets Z axis velocity

3: Sensor APIs-

(i) Accelerometer:

(a) int16_t getX(void)
- Return's X component of acceleration.

(b) int16_t getY(void)
- Return's Y component of acceleration.

(c) int16_t getZ(void)
- Return's Z component of acceleration.

(d) int32_t getNetAcc(void)
- Return's net acceleration.

Note- Unit: cm/sec^2.

(ii) Gyroscope:

(a) int16_t getX(void)

- Return's angular rate about Pluto's X-axis.

(b) int16_t getY(void)

- Return's angular rate about Pluto's Y-axis.

(c) int16_t getZ(void)

- Return's angular rate about Pluto's Z-axis.

Note- Unit: degrees/sec

(iii) Magnetometer:

(a) int16_t getX(void)

- Return's X component of magnetic field.

(b) int16_t getY(void)

- Return's Y component of magnetic field.

(c) int16_t getZ(void)

- Return's Z component of magnetic field.

Note- Unit: microTesla

(iv) Barometer:

(a) int32_t getPressure(void)

- Return's barometer's pressure (mbar) reading.

(b) int16_t getTemperature(void)

- Return's barometer's temperature reading.
Note- Returns temperature in multiple of 0.01degC i.e. 30degC = 3000.

Usage: To access Sensor APIs you have to use predefined respective sensors object 'Accelerometer', 'Gyroscope,' Magnetometer,' Barometer.

Examples: 1- Accelerometer. getNetAcc (); //get net acceleration

2- Barometer. getPressure(); //get pressure

4: Motor APIs-

(i) bool reverseMotorInit(void) //Experimental

- **Initialized the inner (M1-M4) motors having reversibility.**

(ii) void setReferenceFrame(reference_frame_e frame) //Experimental

- **Set Pluto's reference frame for angle calculation**
- **D_FRONT upward facing reference**
- **D_BACK downward facing reference**

(iii) void set(motor_e motor,int16_t value)

- **Set value for motor PWM, allowed values from 1000-2000.**
- **At 1000 motor will stop.**

(iv) void setDirection(motor_e motor,motor_direction_e direction)

- **Set motors rotation direction.**
- **FORWARD for normal configured rotation.**
- **BACKWARD for opposite of configured rotation.**

Note: Pluto-X has total 8 motors(M1-M8); by default outer 4 motors(M5-M8) are configured for Quad and remaing 4(M1-M4
) are available as spare for extra usage; In reverse Mode inner 4(M1-M4) gets configured

for Quad but remaing outer 4(M5-M8) are not available as spare for extra usage; only inner 4 motors(M1-M4) can change direction.

Usage: To access Motor APIs you have to use predefined object 'Motor'.

Examples: 1- Motor.set(M6, 1300) //set M6 to 1300

5: Servo APIs-

(i) void set(servo_e servo,int16_t value)

- **Set value for servo PWM, allowed values from 1000-2000.**
- **At 1500 is mid value.**

Note: Pluto-X currently has one servo(S1) configured on pin 2 on Unibus; As its temporary feature most likely will change in future.

Usage: To access Servo APIs you have to use predefined object 'Servo'.

Examples: 1- Servo.set(S1,1300) //set S1 to 1300

6: Xshield APIs-

(i) void init()

- **Initialise Xshield.**

(ii) void startRanging()

- **Starts laser ranging.**

(iii) void startRanging()void activateAlthold(bool activate)

- **Stops laser ranging.**

(iv) uint16_t getRange(laser_e laser)

- **Return's true if Pluto is in althold mode else false on throttle mod.**

Usage: To access Xshield APIs you have to use predefined object 'Xshield.

Examples: 1- Xshield.getRange(LEFT) //get range from left sensor

7: Angle APIs-

(i) int16_t get(inclination_t angle)

- **Return's current angle (AG_ROLL, AG_PITCH, AG_YAW) value.**

Note- Returns angle (AG_ROLL, AG_PITCH) in multiple of 0.1 degree i.e. 120 degree=1200 and angle(AG_YAW) in degree(0-360).

Usage: To access Angle APIs you have to use predefined object 'Angle'.

Examples: 1- Angle.get(ROLL) //get Pluto's current roll angle

8: LED APIs-

(i) void ledOp(led_e name, led_state_e state)

-set LED (L_LEFT,L_MID,L_RIGHT, right) to a particular state (ON, OFF, TOGGLE).

Usage: Use Function directly.

Example: ledOp(L_LEFT, ON); // on left LED

9: Print APIs-

(i) void monitor(String tag, int32_t number)

- **Print's integer number with tag on Pluto Monitor on Cygnus IDE with 100 ms frequency.**

(ii) void monitor (String tag, double number, uint8_t precision)

- **Print's float number with tag on Pluto Monitor on Cygnus IDE with 100 ms frequency.**

(iii) void monitor (String message)

- **Print's string message on Pluto Monitor on Cygnus IDE with 100 ms frequency.**

(iv) void redGraph(double value, uint8_t precision)

- **Plot value on Red Graph of Pluto Graphs on Cygnus IDE.**

(v) void blueGraph(double value, uint8_t precision)

- **Plot's value on Blue Graph of Pluto Graphs on Cygnus IDE.**

(vi) void greenGraph(double value, uint8_t precision)

- **Plot's value on Green Graph of Pluto Graphs on Cygnus IDE.**

Usage: To access Print APIs you have to use predefined object 'Print'.

Example: 1- Print.monitor("TAG", 34); // print 34 with associated TAG message

2- Print.redGraph(myVariable, 5); // print myVariable at RED GRAPH

10: Utilities-

(i) uint32_t microsT(void)

- **Return's current system time in microseconds.**

(ii) bool Timer::start(uint32_t time)

- **Start's the timer and returns true after every provided time(in milliseconds)**

Note- Allowed time range for Timer is 20ms to 5000ms, Timer is continues one.

Usage: Use microsT(void) directly, to use Timer create Timer object and access the function.

Examples: 1- Timer printTimer // creating Timer object

2- printTimer.start(200) // starting printTimer with 200 milliseconds bound