

CoDrone LINK

Communication

Bluetooth Low Energy

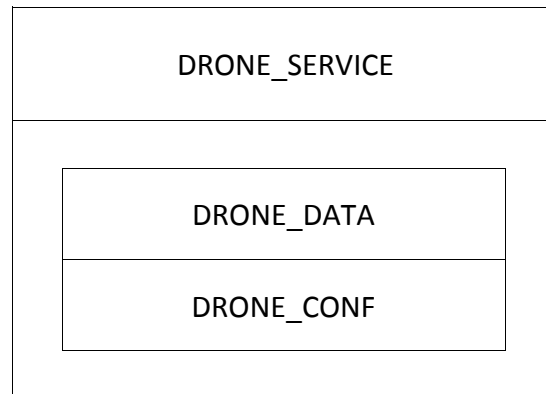
Updated in April 14th, 2016

Contents

1. Introduction
2. Data Link
3. Data Structure

1. Introduction

CoDrone uses Bluetooth Low Energy (BLE) technology.



Name	UUID	Purpose
DRONE_SERVICE	"C320DF00-7891-11E5-8BCF-FEFF819CDC9F"	Service
DRONE_DATA	"C320DF01-7891-11E5-8BCF-FEFF819CDC9F"	CoDrone -> Device(Notify)
DRONE_CONF	"C320DF02-7891-11E5-8BCF-FEFF819CDC9F"	Device -> CoDrone (Write)

2. Data Link

CoDrone takes data from a device through **DRONE_CONF**.

CoDrone sends data to a device through **DRONE_DATA**.

<The array of **DRONE_DATA** and **DRONE_CONF**>

1	2	...	n-1	n
DataType	Data (1)	Data (2)	Data (n-1)	Data (n)

The length of data is not fixed, but the maximum length is 20 bytes.

We recommend to set a data transfer cycle of 50 ms for Android and 100ms for iOS.

When CoDrone is asked to send a particular data, it sends the date required.

When CoDrone takes normal commands, it sends **Ack** back for response.

However, when it takes control commands for moving, it doesn't send **Ack** back.

If you want to receive data from CoDrone to your device, you need to enable "notification" in your application.

<DataType>

```

enum DataType
{
    None = 0,

    // system information
    Ping,                ///< check communication(reserved)
    Ack,                 ///< response to receiving data
    Error,               ///< error(reserved)
    Request,             ///< data request
    Passcode,            ///< reset password for pairing

    // control, command
    Control = 0x10,      ///< control
    Command,             ///< command
    Command2,            ///< multiple command (command 1, 2)
    Command3,            ///< multiple command (command 1, 2, 3)

    // LED
    LedMode = 0x20,      ///< set single LED mode
    LedMode2,            ///< set double LED mode
    LedModeCommand,      ///< LED mode, command
    LedModeCommandIr,    ///< LED mode, command, IR data transfer
    LedModeColor,        ///< LED mode, set single RGB color respectively
    LedModeColor2,       ///< LED mode, set double RGB color respectively

    LedEvent,            ///< single LED event
    LedEvent2,           ///< double LED event
    LedEventCommand,     ///< LED event, command
    LedEventCommandIr,   ///< LED event, command, IR data transfer
    LedEventColor,       ///< LED event, set single RGB color respectively
    LedEventColor2,      ///< LED event, set double RGB color respectively

    LedModeDefaultColor, ///< LED default mode, set single RGB color respectively
    LedModeDefaultColor2, ///< LED default mode, set double RGB color respectively

    // drone condition
    Address = 0x30,      ///< IEEE address
    State,               ///< state(flight mode, coordinate, battery)
    Attitude,            ///< attitude
    GyroBias,            ///< gyro bias
    TrimAll,             ///< fine adjustment
    TrimFlight,          ///< fine adjustment in flight mode
    TrimDrive,           ///< fine adjustment in drive mode

    // Sensor
    ImuRawAndAngle = 0x50, ///< IMU raw data, Angle
    Pressure,            ///< pressure sensor data
    ImageFlow,           ///< optical flow sensor
    Button,              ///< button input
    Battery,             ///< battery
    Motor,               ///< motor control input vales
    Temperature,         ///< temperature

    // data transfer
    IrMessage = 0x40,    ///< IR data transfer

    EndOfType
}

```

3. Data Structure

```
typedef int32_t      s32;
typedef int16_t     s16;
typedef int8_t      s8;

typedef uint32_t     u32;
typedef uint16_t    u16;
typedef uint8_t     u8;
```

3.1. Base

3.1.1. CommandBase

```
struct CommandBase
{
    u8      commandType;    ///< command type
    u8      option;         ///< option for command(use the defined value in System.h)
};
```

CommandBase is a basic structure to ask CoDrone for something, using in structures of **Command**, **Command2**, **Command3**, **LedModeCommand**, **LedModeCommandIr**, **LedEventCommand**, **LedEventCommandIr**

3.1.1.1. CommandType

```
enum CommandType
{
    None = 0,          ///< no event

    // setting
    ModePetrone = 0x10, ///< change petrone mode

    // control
    Coordinate = 0x20,  ///< change coordinate
    Trim,              ///< change fine adjustment value
    FlightEvent,        ///< execute flight event
    DriveEvent,         ///< execute drive event
    Stop,               ///< stop

    ResetHeading = 0x50, ///< Reset heading
    ClearGyroBiasAndTrim, ///< Reset gyrobias and fine adjustment value

    // connection
    PairingActivate = 0x80, ///< enable pairing
    PairingDeactivate,      ///< disable pairing
    TerminateConnection,    ///< connection terminates

    // request
    Request = 0x90,      ///< request data

    EndOfType
};
```

3.1.1.2. Option

Please check the option value of **ModePetrone**, **Coordinate**, **Trim**, **FlightEvent** or **DriveEvent** in **CommandType**.

3.1.1.2.1. ModePetrone

```
{
    None = 0,

    Flight = 0x10,          ///< flight mode with guards
    FlightNoGuard,          ///< flight mode without guards
    FlightFPV,              ///< flight mode (FPV)

    Drive = 0x20,           ///< drive mode
    DriveFPV,               ///< drive mode(FPV)

    Test = 0x30,            ///< test mode

    EndOfType
};
```

3.1.1.2.2. Coordinate

```
enum Coordinate
{
    None = 0,

    Absolute,               ///< fixed coordinate
    Relative,               ///< relative coordinate

    EndOfType
};
```

3.1.1.2.3. Trim

```
enum Trim
{
    None = 0,

    RollIncrease,           ///< Roll up
    RollDecrease,           ///< Roll down
    PitchIncrease,          ///< Pitch up
    PitchDecrease,          ///< Pitch down
    YawIncrease,            ///< Yaw up
    YawDecrease,            ///< Yaw down
    ThrottleIncrease,       ///< Throttle up
    ThrottleDecrease,       ///< Throttle down

    EndOfType
};
```

3.1.1.2.4. FlightEvent

```
enum Trim
{
    None = 0,

    TakeOff,                ///< takeoff

    FlipFront,              ///< flip
    FlipRear,               ///< flip
    FlipLeft,               ///< flip
    FlipRight,              ///< flip
};
```

```

        Stop,                ///< stop
        Landing,             ///< landing
        Reverse,             ///< turtle turn

        Shot,                ///< motion when shooting
        UnderAttack,         ///< motion under attack
        Square,              ///< square flight
        CircleLeft,          ///< circle flight(left)
        CircleRight,         ///< circle flight(right)
        Rotate180,           ///< rotate 180 degrees

        EndOfType
};

```

3.1.1.2.5. Request

```

enum DataType
{
    // drone condition
    Address = 0x30,          ///< IEEE address
    State,                  ///< state(flight mode, coordinate, battery)
    Attitude,               ///< attitude(Vector)
    GyroBias,               ///< gyro bias(Vector)
    TrimAll,                ///< fine adjustment
    TrimFlight,             ///< fine adjustment in flight mode
    TrimDrive,              ///< fine adjustment in drive mode

    // Sensor
    ImuRawAndAngle = 0x50,  ///< IMU raw data, Angle
    Pressure,               ///< pressure sensor data
    ImageFlow,              ///< optical flow sensor
    Button,                 ///< button input
    Battery,                ///< battery
    Motor,                  ///< motor control input vales
    Temperature,            ///< temperature
};

```

3.1.2. LedModeBase

LedModeBase is a basic structure to change various LED modes.

```

struct LedModeBase
{
    u8 mode;                ///< LED mode
    u8 color;               ///< LED color
    u8 interval;            ///< LED interval
};

```

3.1.2.1. mode

```

enum ModeLight
{
    None = 0,

    EyeNone = 0x10,
    EyeHold,                ///< hold color
    EyeMix,                 ///< change color continuously
    EyeFlicker,             ///< flicker
    EyeFlickerDouble,       ///< flicker
    EyeDimming,             ///< control brightness and dimming

    ArmNone = 0x40,
    ArmHold,                ///< hold color
    ArmMix,                 ///< change color continuously
};

```

```

        ArmFlicker,          ///< flicker
        ArmFlickerDouble,    ///< flicker
        ArmDimming,          ///< control brightness and dimming
        ArmFlow,             ///< front to rear
        ArmFlowReverse,      ///< rear to front

        EndOfType
};

```

3.1.2.2. Color

```

enum Colors
{
    AliceBlue,           AntiqueWhite,      Aqua,
    Aquamarine,          Azure,                Beige,
    Bisque,              Black,                BlendedAlmond,
    Blue,                BlueViolet,           Brown,
    BurlyWood,           CadetBlue,            Chartreuse,
    Chocolate,           Coral,                CornflowerBlue,
    Cornsilk,            Crimson,              Cyan,
    DarkBlue,            DarkCyan,             DarkGoldenRod,
    DarkGray,            DarkGreen,            DarkKhaki,
    DarkMagenta,         DarkOliveGreen,       DarkOrange,
    DarkOrchid,          DarkRed,              DarkSalmon,
    DarkSeaGreen,        DarkSlateBlue,        DarkSlateGray,
    DarkTurquoise,       DarkViolet,           DeepPink,
    DeepSkyBlue,         DimGray,              DodgerBlue,
    FireBrick,           FloralWhite,          ForestGreen,
    Fuchsia,             Gainsboro,            GhostWhite,
    Gold,                GoldenRod,            Gray,
    Green,               GreenYellow,          HoneyDew,
    HotPink,             IndianRed,            Indigo,
    Ivory,               Khaki,                Lavender,
    LavenderBlush,       LawnGreen,            LemonChiffon,
    LightBlue,           LightCoral,           LightCyan,
    LightGoldenRodYellow, LightGray,             LightGreen,
    LightPink,           LightSalmon,          LightSeaGreen,
    LightSkyBlue,        LightSlateGray,       LightSteelBlue,
    LightYellow,         Lime,                 LimeGreen,
    Linen,               Magenta,              Maroon,
    MediumAquaMarine,    MediumBlue,           MediumOrchid,
    MediumPurple,        MediumSeaGreen,       MediumSlateBlue,
    MediumSpringGreen,   MediumTurquoise,      MediumVioletRed,
    MidnightBlue,        MintCream,            MistyRose,
    Moccasin,            NavajoWhite,          Navy,
    OldLace,             Olive,                OliveDrab,
    Orange,              OrangeRed,            Orchid,
    PaleGoldenRod,       PaleGreen,            PaleTurquoise,
    PaleVioletRed,       PapayaWhip,           PeachPuff,
    Peru,                Pink,                 Plum,
    PowderBlue,          Purple,               RebeccaPurple,
    Red,                 RosyBrown,            RoyalBlue,
    SaddleBrown,         Salmon,               SandyBrown,
    SeaGreen,            SeaShell,             Sienna,
    Silver,              SkyBlue,              SlateBlue,
    SlateGray,           Snow,                 SpringGreen,
    SteelBlue,           Tan,                  Teal,
    Thistle,             Tomato,               Turquoise,
    Violet,              Wheat,                White,
    WhiteSmoke,          Yellow,               YellowGreen,

    EndOfType
};

```

3.1.2.3. interval

The range of value is 0~255.

3.1.3. ColorBase

ColorBase is a basic structure to control the brightness of R, G or B LED respectively. The range is 0 ~ 255.

```
struct ColorBase
{
    u8 r;          ///< LED Red
    u8 g;          ///< LED Green
    u8 b;          ///< LED Blue
};
```

3.1.4. LedModeColorBase

LedModeColorBase is a basic structure to change the mode of LEDs with controlling the brightness of R, G or B LED respectively.

```
struct LedModeColorBase
{
    u8 mode;        ///< LED mode
    ColorBase color; ///< LED color(R, G, B)
    u8 interval;    ///< LED interval
};
```

3.1.4.1. mode

Refer to **3.1.2.1. mode**

3.1.4.2. color

Refer to **3.1.3. ColorBase**

3.1.4.3. interval

Refer to **3.1.2.3. interval**

3.1.5. LedEventBase

```
struct LedEventBase
{
    u8 event;        ///< LED event
    u8 color;        ///< LED event color
    u8 interval;     ///< LED event interval
    u8 repeat;       ///< LED event repeat
};
```

3.1.5.1. event

Refer to **3.1.2.1. mode**

3.1.5.2. color

Refer to **3.1.2.2. Color**

3.1.5.3. interval

Refer to **3.1.2.3. interval**

3.1.5.4. repeat

The range is 1~255.

3.1.6. LedEventColorBase

```
struct LedEventColorBase
{
    u8          event;          ///< LED event
    ColorBase   color;         ///< LED event color(R, G, B)
    u8          interval;      ///< LED event interval
    u8          repeat;        ///< LED event repeat
};
```

3.1.6.1. event

Refer to **3.1.2.1. mode**

3.1.6.2. color

Refer to **3.1.3. ColorBase**

3.1.6.3. interval

Refer to **3.1.2.3. interval**

3.1.6.4. repeat

The range is 1~255.

3.1.7. MotorBase

```
struct MotorBase
{
    s16         forward;
    s16         reverse;
};
```

3.2. Structures

3.2.1. Ack

Ack is a response data when CoDrone receives commends except data of Flight/Drive control command.

```
struct Ack
{
    u32         systemTime;    ///< receiving time
    u8          dataType;      ///< type of receiving data
};
```

3.2.2. Request

Request is used when you want to get data from CoDrone.

Refer to **3.1.1.2.5. Request**.

```
struct Request
{
    u8          dataType;    ///< datatype to be requested
};
```

3.2.3. Control

Control is information about CODRONE's movement.

```
struct Control
{
    s8          roll;        ///< Roll(left/right)
    s8          pitch;       ///< Pitch(front/rear)
    s8          yaw;         ///< Yaw(heading)
    s8          throttle;    ///< Throttle(up/down)
};
```

Name	Type	Range	Note
roll	s8	-100 ~ 100	Left: - / Right: +
pitch	s8	-100 ~ 100	Rear: - / Front: +
yaw	s8	-100 ~ 100	Heading Counterclockwise: - / Clockwise: +
throttle	s8	-100 ~ 100	Down: - / UP: +

3.2.4. Command

Refer to **3.1.1. CommandBase**.

```
struct Command
{
    CommandBase  command;
};
```

3.2.5. Command2

Refer to **3.1.1. CommandBase**.

```
struct Command2
{
    CommandBase  command1;
    CommandBase  command2;
};
```

3.2.6. Command3

Refer to **3.1.1. CommandBase**.

```
struct Command3
{
    CommandBase  command1;
    CommandBase  command2;
    CommandBase  command3;
};
```

3.2.7. LedMode

Refer to **3.1.2. LedModeBase**.

```
struct LedMode
{
    LedModeBase    ledMode;
};
```

3.2.8. LedMode2

Refer to **3.1.2. LedModeBase**.

```
struct LedMode2
{
    LedModeBase    ledMode1;
    LedModeBase    ledMode2;
};
```

3.2.9. LedModeCommand

Refer to **3.1.2. LedModeBase** and **3.1.1. CommandBase**.

```
struct LedModeCommand
{
    LedModeBase    ledMode;
    CommandBase    command;
};
```

3.2.10. LedModeCommandIr

Refer to **3.1.2. LedModeBase** and **3.1.1. CommandBase**.

The size of **irData** is 32bits(4bytes).

Control Device ="irData" => CODRONE(IR Transmitter) => another CODRONE(IR Receiver)
="IrMessage"=> another Control Device

```
struct LedModeCommandIr
{
    LedModeBase    ledMode;
    CommandBase    command;
    u32            irData;
};
```

3.2.11. LedModeColor

Refer to **3.1.4. LedModeColorBase**.

```
struct LedModeColor
{
    LedModeColorBase    ledModeColor;
};
```

3.2.12. LedModeColor2

Refer to **3.1.4. LedModeColorBase**.

```
struct LedModeColor2
{
    LedModeColorBase    ledModeColor1;
    LedModeColorBase    ledModeColor2;
};
```

3.2.13. LedEvent

Refer to **3.1.5. LedEventBase**.

```
struct LedEvent
{
    LedEventBase    ledEvent;
};
```

3.2.14. LedEvent2

Refer to **3.1.5. LedEventBase**.

```
struct LedEvent2
{
    LedEventBase    ledEvent1;
    LedEventBase    ledEvent2;
};
```

3.2.15. LedEventCommand

Refer to **3.1.5. LedEventBase** and **3.1.1. CommandBase**.

```
struct LedEventCommand
{
    LedEventBase    ledEvent;
    CommandBase     command;
};
```

3.2.16. LedEventCommandIr

Refer to **3.1.5. LedEventBase** and **3.1.1. CommandBase**.

```
struct LedEventCommandIr
{
    LedEventBase    ledEvent;
    CommandBase     command;
    u32             irData;
};
```

3.2.17. LedEventColor

Refer to **3.1.6. LedEventColorBase**.

```
struct LedEventColor
{
    LedEventColorBase    ledEventColor;
};
```

3.2.18. LedEventColor2

Refer to **3.1.6. LedEventColorBase**.

```
struct LedEventColor2
{
    LedEventColorBase    ledEventColor1;
    LedEventColorBase    ledEventColor2;
};
```

3.2.19. LedModeDefaultColor

Refer to **3.1.4. LedModeColorBase**.

The data set by **LedModeColorBase** is memorized in flash memory. When you start drone, it comes as a default value.

```
struct LedModeDefaultColor
```

```

{
    LedModeColorBase    ledModeColor;
};

```

3.2.20. LedModeDefaultColor2

Refer to **3.1.4. LedModeColorBase**.

```

struct LedModeDefaultColor2
{
    LedModeColorBase    ledModeColor1;
    LedModeColorBase    ledModeColor2;
};

```

3.2.21. Address

CODRONE's MAC Address.

```

struct Address
{
    u8    address[6];
};

```

3.2.22. State

```

struct State
{
    u8    modePetrone;    ///< CoDrone mode

    u8    modeVehicle;    ///< movement mode
    u8    modeFlight;    ///< flight mode
    u8    modeDrive;    ///< drive mode

    u8    sensorOrientation;    ///< sensor orientation
    u8    coordinate;    ///< coordinate
    u8    battery;    ///< battery check (0 ~ 100)
};

```

3.2.22.1. ModePetrone

Refer to **3.1.1.2.1. ModePetrone**

3.2.22.2. ModeVehicle

```

enum ModeVehicle
{
    None = 0,

    Boot,    ///< booting
    Wait,    ///< wait for connection

    Ready,    ///< ready to start

    Running,    ///< main course

    Update,    ///< firmware update
    UpdateComplete,    ///

```

3.2.22.3. ModeFlight

```

enum ModeFlight
{
    None = 0,

    Ready,                ///< ready to fly

    TakeOff,              ///< take off
    Flight,               ///< flight
    Flip,                 ///< flip
    Stop,                 ///< emergency stop
    Landing,              ///< landing
    Reverse,              ///< turtle turn

    Accident,             ///< convert into ready mode
    Error,                ///< error

    EndOfType
};

```

3.2.22.4. ModeDrive

```

enum ModeDrive
{
    None = 0,

    Ready,                ///< ready to drive

    Start,                ///< start
    Drive,                ///< drive
    Stop,                 ///< emergency stop

    Accident,             ///< convert into ready mode
    Error,                ///< error

    EndOfType
};

```

3.2.22.5. SensorOrientation

```

enum SensorOrientation
{
    None = 0,

    Normal,               ///< OK
    ReverseStart,         ///< start turtle turn
    Reverse,              ///< finish

    EndOfType
};

```

3.2.22.6. Coordinate

Refer to **3.1.1.2.2. Coordinate**

3.2.22.7. battery

The range is 0~100.

3.2.23. Attitude

CODRONE's attitude angles(degree)

Struct Attitude

{	s16	roll;
	s16	pitch;
	s16	yaw;
};		

<angle data(degree)>

Name	Type	Range	Note
roll	s16	-180 ~ 180	tilting angle of left/right
pitch	s16	-180 ~ 180	tilting angle of front/rear
yaw	s16	0 ~ 360	Heading angle

<raw data>

Name	Type	Range	Note
roll	s16	-32768 ~ 32767	slope value of left/right
pitch	s16	-32768 ~ 32767	slope value of front/rear
yaw	s16	-32768 ~ 32767	Heading value

3.2.24. GyroBias

Checking the gyro bias value. Refer to **3.2.23. Attitude**.

3.2.25. TrimAll

Fine adjustment

struct TrimAll
{
TrimFlight flight;
TrimDrive drive;
};

3.2.26. TrimFlight

Fine adjustment in flight mode. The range is -100~100.

struct TrimFlight
{
s16 roll;
s16 pitch;
s16 yaw;
s16 throttle;
};

3.2.27. TrimDrive

Fine adjustment in drive mode. The range is -100~100.

struct TrimDrive
{
s16 wheel;
};

3.2.28. IrMessage


```

struct IrMessage
{
    u8          direction;
    u32          irData;
};

```

3.2.28.1. Direction

IR receivers are in front and rear of CODRONE.

```

enum Direction
{
    None = 0,
    Left,
    Front,
    Right,
    Rear,
    EndOfType
};

```

3.2.28.2. irData

IR transfer data. The size is 4 byte.

3.2.29. ImuRawAndAngle

```

struct ImuRawAndAngle
{
    s16          accX;          ///< raw data
    s16          accY;          ///< raw data
    s16          accZ;          ///< raw data
    s16          gyroRoll;      ///< raw data
    s16          gyroPitch;     ///< raw data
    s16          gyroYaw;       ///< raw data
    s16          angleRoll;     ///< angle(roll)
    s16          anglePitch;    ///< angle(pitch)
    s16          angleYaw;      ///< angle(yaw)
};

```

3.2.30. Pressure

```

struct Pressure
{
    s32          d1;
    s32          d2;
    s32          temperature;
    s32          pressure;
};

```

3.2.31. ImageFlow

```

struct ImageFlow
{
    s32          fVelocitySumX;
    s32          fVelocitySumY;
};

```

3.2.32. Button

```

struct Button
{

```

```
};          u8          button;
```

3.2.33. Battery

```
struct Battery
{
    s16          v30;
    s16          v33;
    s16          gradient;
    s16          yIntercept;
    u8           flagBatteryCalibration;

    s32          batteryRaw;
    s8           batteryPercent;

    s16          voltage;
};
```

3.2.34. Motor

```
struct Motor
{
    MotorBase motor[4];
};
```

3.2.35. Temperature

```
struct Temperature
{
    s32          imu;
    s32          pressure;
};
```