```java
package com.MAVLink.Messages;

import java.util.ArrayList;

import com.MAVLink.Messages.enums.MAV_CMD;

public enum ApmCommands {

        CMD_NAV_WAYPOINT ("Waypoint",MAV_CMD.MAV_CMD_NAV_WAYPOINT,CommandType.NAVIGATION), /* Navigate to MISSION. |Hold time in decimal seconds. (ignored by fixed wing, time to stay at MISSION for rotary wing)| Acceptance radius in meters (if the sphere with this radius is hit, the MISSION counts as reached)| 0 to pass through the WP, if > 0 radius in meters to pass by WP. Positive value for clockwise orbit, negative value for counter-clockwise orbit. Allows trajectory control.| Desired yaw angle at MISSION (rotary wing)| Latitude| Longitude| Altitude|  */
        CMD_NAV_LOITER_UNLIM("Loiter",MAV_CMD.MAV_CMD_NAV_LOITER_UNLIM ,CommandType.NAVIGATION), /* Loiter around this MISSION an unlimited amount of time |Empty| Empty| Radius around MISSION, in meters. If positive loiter clockwise, else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        CMD_NAV_LOITER_TURNS("LoiterN",MAV_CMD.MAV_CMD_NAV_LOITER_TURNS ,CommandType.NAVIGATION), /* Loiter around this MISSION for X turns |Turns| Empty| Radius around MISSION, in meters. If positive loiter clockwise, else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        CMD_NAV_LOITER_TIME ("LoiterT",MAV_CMD.MAV_CMD_NAV_LOITER_TIME ,CommandType.NAVIGATION), /* Loiter around this MISSION for X seconds |Seconds (decimal)| Empty| Radius around MISSION, in meters. If positive loiter clockwise, else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        CMD_NAV_RETURN_TO_LAUNCH("RTL",MAV_CMD.MAV_CMD_NAV_RETURN_TO_LAUNCH,CommandType.COMMAND), /* Return to launch location |Empty| Empty| Empty| Empty| Empty| Empty| Empty|  */
        CMD_NAV_LAND("Land",MAV_CMD.MAV_CMD_NAV_LAND,CommandType.NAVIGATION), /* Land at location |Empty| Empty| Empty| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        CMD_NAV_TAKEOFF("Takeoff",MAV_CMD.MAV_CMD_NAV_TAKEOFF,CommandType.NAVIGATION), /* Takeoff from ground / hand |Minimum pitch (if airspeed sensor present), desired pitch without sensor| Empty| Empty| Yaw angle (if magnetometer present), ignored without magnetometer| Latitude| Longitude| Altitude|  */
        CMD_NAV_ROI("ROI",MAV_CMD.MAV_CMD_NAV_ROI,CommandType.COMMAND_WITH_TARGET), /* Sets the region of interest (ROI) for a sensor set or the vehicle itself. This can then be used by the vehicles control system to control the vehicle attitude and the attitude of various sensors such as cameras. |Region of intereset mode. (see MAV_ROI enum)| MISSION index/ target ID. (see MAV_ROI enum)| ROI index (allows a vehicle to manage multiple ROI's)| Empty| x the location of the fixed ROI (see MAV_FRAME)| y| z|  */
        CMD_NAV_PATHPLANNING("Path",MAV_CMD.MAV_CMD_NAV_PATHPLANNING,CommandType.COMMAND), /* Control autonomous path planning on the MAV. |0: Disable local obstacle avoidance / local path planning (without resetting map), 1: Enable local path planning, 2: Enable and reset local path planning| 0: Disable full path planning (without resetting map), 1: Enable, 2: Enable and reset map/occupancy grid, 3: Enable and reset planned route, but not occupancy grid| Empty| Yaw angle at goal, in compass degrees, [0..360]| Latitude/X of goal| Longitude/Y of goal| Altitude/Z of goal|  */
        CMD_DO_JUMP("Do Jump",MAV_CMD.MAV_CMD_DO_JUMP,CommandType.COMMAND), /* Jump to the desired command in the mission list.  Repeat this action only the specified number of times |Sequence number| Repeat count| Empty| Empty| Empty| Empty| Empty|  */
        CMD_DO_SET_HOME("Set Home",MAV_CMD.MAV_CMD_DO_SET_HOME,CommandType.COMMAND_WITH_TARGET), /*       Changes the home location either to the current location or a specified location.| Use current (1=use current location, 0=use specified location) | Empty | Empty | Empty | Latitude | Longitude | Altitude*/
        CMD_DO_CHANGE_SPEED("Set Speed",MAV_CMD.MAV_CMD_DO_CHANGE_SPEED,CommandType.COMMAND), /*  Change speed and/or throttle set points.| Speed type (0=Airspeed, 1=Ground Speed) | Speed (m/s, -1 indicates no change)  | Throttle ( Percent, -1 indicates no change) | Empty | Empty | Empty | Empty*/
        CMD_CONDITION_CHANGE_ALT("Set Alt",MAV_CMD.MAV_CMD_CONDITION_CHANGE_ALT,CommandType.NAVIGATION),/*Ascend/descend at rate. Delay mission state machine until desired altitude reached.| Descent / Ascend rate (m/s) |       Empty | Empty | Empty | Empty | Empty | Finish Altitude | */
        CMD_CONDITION_DISTANCE("Set Distance",MAV_CMD.MAV_CMD_CONDITION_DISTANCE,CommandType.COMMAND),    /*Delay mission state machine until within desired distance of next NAV point.| Distance (meters) | Empty | Empty | Empty | Empty | Empty | Empty*/
        CMD_CONDITION_YAW("Yaw to",MAV_CMD.MAV_CMD_CONDITION_YAW,CommandType.COMMAND),  /* Yaw to heading while executing next waypoint.  |  Target angle: [0-360], 0 is north. |    speed during yaw change:[deg per second] |      direction: negative: counter clockwise, positive: clockwise [-1,1] | relative offset or absolute angle: [ 1,0] | Empty| Empty| Empty|  */
//      CMD_DO_SET_RELAY("Set Relay",MAV_CMD.MAV_CMD_DO_SET_RELAY,CommandType.COMMAND),/* Set a relay to a condition.| Relay number       | Setting (1=on, 0=off, others possible depending on system hardware) | Empty | Empty | Empty    | Empty | Empty*/
//      CMD_DO_REPEAT_RELAY("Repeat Relay",MAV_CMD.MAV_CMD_DO_REPEAT_RELAY,CommandType.COMMAND),/*        Cycle a relay on and off for a desired number of cyles with a desired period.    | Relay number  | Cycle count   | Cycle time (seconds, decimal) | Empty | Empty  | Empty | Empty */
        ;

        private final String name;
        private final int arduPilotIntValue;
        private final CommandType commandType;

        ApmCommands(String name, int type, CommandType showOnMap){
                this.name = name;
                this.arduPilotIntValue = type;
                this.commandType = showOnMap;
        }

        public String getName() {
                return name;
        }

        public int getType() {
                return arduPilotIntValue;
        }

        public boolean showOnMap(){
                switch (this.commandType) {
                case COMMAND:
                        return false;
                default:
                case COMMAND_WITH_TARGET:
                case NAVIGATION:
                        return true;
                }
        }

        public boolean isOnFligthPath(){
                switch (this.commandType) {
                default:
                case COMMAND:
                case COMMAND_WITH_TARGET:
                        return false;
                case NAVIGATION:
                        return true;
                }
        }

        public static ApmCommands getCmd(int type) {
                for (ApmCommands mode : ApmCommands.values()) {
                        if (type == mode.getType()) {
                                return mode;
                        }
                }
                return null;
        }

        public static ApmCommands getCmd(String str) {
                for (ApmCommands mode : ApmCommands.values()) {
                        if (str.equals(mode.getName())) {
                                return mode;
                        }
                }
                return null;
        }

        public static ArrayList<String> getNameList() {
                ArrayList<String> list = new ArrayList<String>();

                for (ApmCommands mode : ApmCommands.values()) {
                                list.add(mode.getName());
                }
                return list;
        }


        private enum CommandType{
                NAVIGATION,
                COMMAND,
                COMMAND_WITH_TARGET
        };

}
```

```java
package com.MAVLink.Messages;

import java.util.ArrayList;
import java.util.List;

import com.MAVLink.Messages.enums.MAV_TYPE;

public enum ApmModes {
        FIXED_WING_MANUAL (0,"Manual",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_CIRCLE (1,"Circle",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_STABILIZE (2,"Stabilize",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_TRAINING (3,"Training",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_FLY_BY_WIRE_A (5,"FBW A",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_FLY_BY_WIRE_B (6,"FBW B",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_AUTO (10,"Auto",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_RTL (11,"RTL",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_LOITER (12,"Loiter",MAV_TYPE.MAV_TYPE_FIXED_WING),
        FIXED_WING_GUIDED (15,"Guided",MAV_TYPE.MAV_TYPE_FIXED_WING),

        ROTOR_STABILIZE(0, "Stabilize", MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_ACRO(1,"Acro", MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_ALT_HOLD(2, "Alt Hold",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_AUTO(3, "Auto",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_GUIDED(4, "Guided",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_LOITER(5, "Loiter",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_RTL(6, "RTL",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_CIRCLE(7, "Circle",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_POSITION(8, "Pos Hold",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_LAND(9, "Land",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_TOY(11, "Toy",MAV_TYPE.MAV_TYPE_QUADROTOR),
        ROTOR_TAKEOFF(12, "Takeoff",MAV_TYPE.MAV_TYPE_QUADROTOR),

        ROVER_MANUAL(0, "MANUAL", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
        ROVER_LEARNING(2, "LEARNING", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
        ROVER_STEERING(3, "STEERING", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
        ROVER_HOLD(4, "HOLD", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
        ROVER_AUTO(10, "AUTO", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
        ROVER_RTL(11, "RTL", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
        ROVER_GUIDED(15, "GUIDED", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
        ROVER_INITIALIZING(16, "INITIALIZING", MAV_TYPE.MAV_TYPE_GROUND_ROVER),


        UNKNOWN(-1, "Unknown",0);



        private final int number;
    private final String name;
        private final int type;

        ApmModes(int number,String name, int type){
                this.number = number;
                this.name = name;
                this.type = type;
        }

        public int getNumber() {
                return number;
        }

        public String getName() {
                return name;
        }

        public int getType() {
                return type;
        }

        public static ApmModes getMode(int i, int type) {
                for (ApmModes mode : ApmModes.values()) {
                        if (i == mode.getNumber() & type == mode.getType()) {
                                return mode;
                        }
                }
                return UNKNOWN;
        }

        public static ApmModes getMode(String str, int type) {
                for (ApmModes mode : ApmModes.values()) {
                        if (str.equals(mode.getName()) & type == mode.getType()) {
                                return mode;
                        }
                }
                return UNKNOWN;
        }

        public static List<ApmModes> getModeList(int type) {
                List<ApmModes> modeList = new ArrayList<ApmModes>();

                if (isCopter(type)) {
                        type = MAV_TYPE.MAV_TYPE_QUADROTOR;
                }

                for (ApmModes mode : ApmModes.values()) {
                        if (isValid(mode) & mode.getType() == type) {
                                modeList.add(mode);
                        }
                }
                return modeList;
        }

        public static boolean isValid(ApmModes mode) {
                return mode!=ApmModes.UNKNOWN;
        }


        public static boolean isCopter(int type){
                switch (type) {
                case MAV_TYPE.MAV_TYPE_TRICOPTER:
                case MAV_TYPE.MAV_TYPE_QUADROTOR:
                case MAV_TYPE.MAV_TYPE_HEXAROTOR:
                case MAV_TYPE.MAV_TYPE_OCTOROTOR:
                case MAV_TYPE.MAV_TYPE_HELICOPTER:
                        return true;
                case MAV_TYPE.MAV_TYPE_FIXED_WING:
                default:
                        return false;
                }
        }

}
```

```java
// MESSAGE AHRS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Status of DCM attitude estimator
*/
public class msg_ahrs extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_AHRS = 163;
        public static final int MAVLINK_MSG_LENGTH = 28;
        private static final long serialVersionUID = MAVLINK_MSG_ID_AHRS;


        /**
        * X gyro drift estimate rad/s
        */
        public float omegaIx;
        /**
        * Y gyro drift estimate rad/s
        */
        public float omegaIy;
        /**
        * Z gyro drift estimate rad/s
        */
        public float omegaIz;
        /**
        * average accel_weight
        */
        public float accel_weight;
        /**
        * average renormalisation value
        */
        public float renorm_val;
        /**
        * average error_roll_pitch value
        */
        public float error_rp;
        /**
        * average error_yaw value
        */
        public float error_yaw;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_AHRS;
                packet.payload.putFloat(omegaIx);
                packet.payload.putFloat(omegaIy);
                packet.payload.putFloat(omegaIz);
                packet.payload.putFloat(accel_weight);
                packet.payload.putFloat(renorm_val);
                packet.payload.putFloat(error_rp);
                packet.payload.putFloat(error_yaw);
                return packet;
        }

    /**
     * Decode a ahrs message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            omegaIx = payload.getFloat();
            omegaIy = payload.getFloat();
            omegaIz = payload.getFloat();
            accel_weight = payload.getFloat();
            renorm_val = payload.getFloat();
            error_rp = payload.getFloat();
            error_yaw = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_ahrs(){
        msgid = MAVLINK_MSG_ID_AHRS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_ahrs(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_AHRS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "AHRS");
        //Log.d("MAVLINK_MSG_ID_AHRS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_AHRS -"+" omegaIx:"+omegaIx+" omegaIy:"+omegaIy+" omegaIz:"+omegaIz+" accel_weight:"+accel_weight+" renorm_val:"+renorm_val+" error_rp:"+error_rp+" error_yaw:"+error_yaw+"";
    }
}
```

```java
// MESSAGE AP_ADC PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* raw ADC output
*/
public class msg_ap_adc extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_AP_ADC = 153;
        public static final int MAVLINK_MSG_LENGTH = 12;
        private static final long serialVersionUID = MAVLINK_MSG_ID_AP_ADC;


        /**
        * ADC output 1
        */
        public short adc1;
        /**
        * ADC output 2
        */
        public short adc2;
        /**
        * ADC output 3
        */
        public short adc3;
        /**
        * ADC output 4
        */
        public short adc4;
        /**
        * ADC output 5
        */
        public short adc5;
        /**
        * ADC output 6
        */
        public short adc6;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_AP_ADC;
                packet.payload.putShort(adc1);
                packet.payload.putShort(adc2);
                packet.payload.putShort(adc3);
                packet.payload.putShort(adc4);
                packet.payload.putShort(adc5);
                packet.payload.putShort(adc6);
                return packet;
        }

    /**
     * Decode a ap_adc message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            adc1 = payload.getShort();
            adc2 = payload.getShort();
            adc3 = payload.getShort();
            adc4 = payload.getShort();
            adc5 = payload.getShort();
            adc6 = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_ap_adc(){
        msgid = MAVLINK_MSG_ID_AP_ADC;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_ap_adc(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_AP_ADC;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "AP_ADC");
        //Log.d("MAVLINK_MSG_ID_AP_ADC", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_AP_ADC -"+" adc1:"+adc1+" adc2:"+adc2+" adc3:"+adc3
+" adc4:"+adc4+" adc5:"+adc5+" adc6:"+adc6+"";
    }
}
```

```java
// MESSAGE ATTITUDE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The attitude in the aeronautical frame (right-handed, Z-down, X-front, Y-right).
*/
public class msg_attitude extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_ATTITUDE = 30;
        public static final int MAVLINK_MSG_LENGTH = 28;
        private static final long serialVersionUID = MAVLINK_MSG_ID_ATTITUDE;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * Roll angle (rad, -pi..+pi)
        */
        public float roll;
        /**
        * Pitch angle (rad, -pi..+pi)
        */
        public float pitch;
        /**
        * Yaw angle (rad, -pi..+pi)
        */
        public float yaw;
        /**
        * Roll angular speed (rad/s)
        */
        public float rollspeed;
        /**
        * Pitch angular speed (rad/s)
        */
        public float pitchspeed;
        /**
        * Yaw angular speed (rad/s)
        */
        public float yawspeed;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_ATTITUDE;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                packet.payload.putFloat(rollspeed);
                packet.payload.putFloat(pitchspeed);
                packet.payload.putFloat(yawspeed);
                return packet;
        }

    /**
     * Decode a attitude message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
            rollspeed = payload.getFloat();
            pitchspeed = payload.getFloat();
            yawspeed = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_attitude(){
        msgid = MAVLINK_MSG_ID_ATTITUDE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_attitude(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_ATTITUDE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "ATTITUDE");
        //Log.d("MAVLINK_MSG_ID_ATTITUDE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_ATTITUDE -"+" time_boot_ms:"+time_boot_ms+" roll:"+
roll+" pitch:"+pitch+" yaw:"+yaw+" rollspeed:"+rollspeed+" pitchspeed:"+pitchspeed+" yaws
peed:"+yawspeed+"";
    }
}
```

```java
// MESSAGE ATTITUDE_QUATERNION PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The attitude in the aeronautical frame (right-handed, Z-down, X-front, Y-right), expressed as quaternion.
*/
public class msg_attitude_quaternion extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_ATTITUDE_QUATERNION = 31;
        public static final int MAVLINK_MSG_LENGTH = 32;
        private static final long serialVersionUID = MAVLINK_MSG_ID_ATTITUDE_QUATERNION;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * Quaternion component 1
        */
        public float q1;
        /**
        * Quaternion component 2
        */
        public float q2;
        /**
        * Quaternion component 3
        */
        public float q3;
        /**
        * Quaternion component 4
        */
        public float q4;
        /**
        * Roll angular speed (rad/s)
        */
        public float rollspeed;
        /**
        * Pitch angular speed (rad/s)
        */
        public float pitchspeed;
        /**
        * Yaw angular speed (rad/s)
        */
        public float yawspeed;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_ATTITUDE_QUATERNION;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(q1);
                packet.payload.putFloat(q2);
                packet.payload.putFloat(q3);
                packet.payload.putFloat(q4);
                packet.payload.putFloat(rollspeed);
                packet.payload.putFloat(pitchspeed);
                packet.payload.putFloat(yawspeed);
                return packet;
        }

    /**
     * Decode a attitude_quaternion message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            q1 = payload.getFloat();
            q2 = payload.getFloat();
            q3 = payload.getFloat();
            q4 = payload.getFloat();
            rollspeed = payload.getFloat();
            pitchspeed = payload.getFloat();
            yawspeed = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_attitude_quaternion(){
        msgid = MAVLINK_MSG_ID_ATTITUDE_QUATERNION;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_attitude_quaternion(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_ATTITUDE_QUATERNION;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "ATTITUDE_QUATERNION");
        //Log.d("MAVLINK_MSG_ID_ATTITUDE_QUATERNION", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_ATTITUDE_QUATERNION -"+" time_boot_ms:"+time_boot_ms+" q1:"+q1+" q2:"+q2+" q3:"+q3+" q4:"+q4+" rollspeed:"+rollspeed+" pitchspeed:"+pitchspeed+" yawspeed:"+yawspeed+"";
    }
}
```

```java
// MESSAGE AUTH_KEY PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Emit an encrypted signature / key identifying this system. PLEASE NOTE: This protocol has been kept simple, so transmitting the key requires an encrypted channel for true safety.
*/
public class msg_auth_key extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_AUTH_KEY = 7;
        public static final int MAVLINK_MSG_LENGTH = 32;
        private static final long serialVersionUID = MAVLINK_MSG_ID_AUTH_KEY;


        /**
        * key
        */
        public byte key[] = new byte[32];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_AUTH_KEY;
                 for (int i = 0; i < key.length; i++) {
                        packet.payload.putByte(key[i]);
            }
                return packet;
        }

    /**
     * Decode a auth_key message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
             for (int i = 0; i < key.length; i++) {
                        key[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_auth_key(){
        msgid = MAVLINK_MSG_ID_AUTH_KEY;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_auth_key(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_AUTH_KEY;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "AUTH_KEY");
        //Log.d("MAVLINK_MSG_ID_AUTH_KEY", toString());
    }

 /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setKey(String str) {
      int len = Math.min(str.length(), 32);
      for (int i=0; i<len; i++) {
        key[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<32; i++) {                      // padding for the rest of the buffer
        key[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getKey() {
                String result = "";
                for (int i = 0; i < 32; i++) {
                        if (key[i] != 0)
                                result = result + (char) key[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_AUTH_KEY -"+" key:"+key+"";
    }
}
```

```java
// MESSAGE BATTERY_STATUS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Transmitte battery informations for a accu pack.
*/
public class msg_battery_status extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_BATTERY_STATUS = 147;
        public static final int MAVLINK_MSG_LENGTH = 16;
        private static final long serialVersionUID = MAVLINK_MSG_ID_BATTERY_STATUS;


        /**
        * Battery voltage of cell 1, in millivolts (1 = 1 millivolt)
        */
        public short voltage_cell_1;
        /**
        * Battery voltage of cell 2, in millivolts (1 = 1 millivolt), -1: no cell
        */
        public short voltage_cell_2;
        /**
        * Battery voltage of cell 3, in millivolts (1 = 1 millivolt), -1: no cell
        */
        public short voltage_cell_3;
        /**
        * Battery voltage of cell 4, in millivolts (1 = 1 millivolt), -1: no cell
        */
        public short voltage_cell_4;
        /**
        * Battery voltage of cell 5, in millivolts (1 = 1 millivolt), -1: no cell
        */
        public short voltage_cell_5;
        /**
        * Battery voltage of cell 6, in millivolts (1 = 1 millivolt), -1: no cell
        */
        public short voltage_cell_6;
        /**
        * Battery current, in 10*milliamperes (1 = 10 milliampere), -1: autopilot does not measure the current
        */
        public short current_battery;
        /**
        * Accupack ID
        */
        public byte accu_id;
        /**
        * Remaining battery energy: (0%: 0, 100%: 100), -1: autopilot does not estimate the remaining battery
        */
        public byte battery_remaining;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_BATTERY_STATUS;
                packet.payload.putShort(voltage_cell_1);
                packet.payload.putShort(voltage_cell_2);
                packet.payload.putShort(voltage_cell_3);
                packet.payload.putShort(voltage_cell_4);
                packet.payload.putShort(voltage_cell_5);
                packet.payload.putShort(voltage_cell_6);
                packet.payload.putShort(current_battery);
                packet.payload.putByte(accu_id);
                packet.payload.putByte(battery_remaining);
                return packet;
        }

    /**
     * Decode a battery_status message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            voltage_cell_1 = payload.getShort();
            voltage_cell_2 = payload.getShort();
            voltage_cell_3 = payload.getShort();
            voltage_cell_4 = payload.getShort();
            voltage_cell_5 = payload.getShort();
            voltage_cell_6 = payload.getShort();
            current_battery = payload.getShort();
            accu_id = payload.getByte();
            battery_remaining = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_battery_status(){
        msgid = MAVLINK_MSG_ID_BATTERY_STATUS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_battery_status(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_BATTERY_STATUS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "BATTERY_STATUS");
        //Log.d("MAVLINK_MSG_ID_BATTERY_STATUS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_BATTERY_STATUS -"+" voltage_cell_1:"+voltage_cell_1+" voltage_cell_2:"+voltage_cell_2+" voltage_cell_3:"+voltage_cell_3+" voltage_cell_4:"+voltage_cell_4+" voltage_cell_5:"+voltage_cell_5+" voltage_cell_6:"+voltage_cell_6+" current_battery:"+current_battery+" accu_id:"+accu_id+" battery_remaining:"+battery_remaining+"";
    }
}
```

```java
// MESSAGE CHANGE_OPERATOR_CONTROL PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Request to control this MAV
*/
public class msg_change_operator_control extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL = 5;
        public static final int MAVLINK_MSG_LENGTH = 28;
        private static final long serialVersionUID = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;


        /**
        * System the GCS requests control for
        */
        public byte target_system;
        /**
        * 0: request control of this MAV, 1: Release control of this MAV
        */
        public byte control_request;
        /**
        * 0: key as plaintext, 1-255: future, different hashing/encryption variants. The GCS should in general use the safest mode possible initially and then gradually move down the encryption level if it gets a NACK message indicating an encryption mismatch.
        */
        public byte version;
        /**
        * Password / Key, depending on version plaintext or encrypted. 25 or less characters, NULL terminated. The characters may involve A-Z, a-z, 0-9, and "!?,.-"
        */
        public byte passkey[] = new byte[25];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;
                packet.payload.putByte(target_system);
                packet.payload.putByte(control_request);
                packet.payload.putByte(version);
                 for (int i = 0; i < passkey.length; i++) {
                        packet.payload.putByte(passkey[i]);
            }
                return packet;
        }

    /**
     * Decode a change_operator_control message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            target_system = payload.getByte();
            control_request = payload.getByte();
            version = payload.getByte();
             for (int i = 0; i < passkey.length; i++) {
                        passkey[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_change_operator_control(){
        msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_change_operator_control(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "CHANGE_OPERATOR_CONTROL");
        //Log.d("MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL", toString());
    }

       /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setPasskey(String str) {
      int len = Math.min(str.length(), 25);
      for (int i=0; i<len; i++) {
        passkey[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<25; i++) {                      // padding for the rest of the buffer
        passkey[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getPasskey() {
                String result = "";
                for (int i = 0; i < 25; i++) {
                        if (passkey[i] != 0)
                                result = result + (char) passkey[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL -"+" target_system:"+target_system+" control_request:"+control_request+" version:"+version+" passkey:"+passkey+"";
    }
}
```

```java
// MESSAGE CHANGE_OPERATOR_CONTROL_ACK PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Accept / deny control of this MAV
*/
public class msg_change_operator_control_ack extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK = 6;
        public static final int MAVLINK_MSG_LENGTH = 3;
        private static final long serialVersionUID = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;


        /**
        * ID of the GCS this message
        */
        public byte gcs_system_id;
        /**
        * 0: request control of this MAV, 1: Release control of this MAV
        */
        public byte control_request;
        /**
        * 0: ACK, 1: NACK: Wrong passkey, 2: NACK: Unsupported passkey encryption method, 3: NACK: Already under control
        */
        public byte ack;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;
                packet.payload.putByte(gcs_system_id);
                packet.payload.putByte(control_request);
                packet.payload.putByte(ack);
                return packet;
        }

    /**
     * Decode a change_operator_control_ack message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            gcs_system_id = payload.getByte();
            control_request = payload.getByte();
            ack = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_change_operator_control_ack(){
        msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_change_operator_control_ack(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "CHANGE_OPERATOR_CONTROL_ACK");
        //Log.d("MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK -"+" gcs_system_id:"+gcs_system_id+" control_request:"+control_request+" ack:"+ack+"";
    }
}
```

```java
// MESSAGE COMMAND_ACK PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Report status of a command. Includes feedback wether the command was executed.
*/
public class msg_command_ack extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_COMMAND_ACK = 77;
        public static final int MAVLINK_MSG_LENGTH = 3;
        private static final long serialVersionUID = MAVLINK_MSG_ID_COMMAND_ACK;


        /**
        * Command ID, as defined by MAV_CMD enum.
        */
        public short command;
        /**
        * See MAV_RESULT enum
        */
        public byte result;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_COMMAND_ACK;
                packet.payload.putShort(command);
                packet.payload.putByte(result);
                return packet;
        }

    /**
     * Decode a command_ack message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            command = payload.getShort();
            result = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_command_ack(){
        msgid = MAVLINK_MSG_ID_COMMAND_ACK;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_command_ack(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_COMMAND_ACK;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "COMMAND_ACK");
        //Log.d("MAVLINK_MSG_ID_COMMAND_ACK", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_COMMAND_ACK -"+" command:"+command+" result:"+result+"";
    }
}
```

```java
// MESSAGE COMMAND_LONG PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Send a command with up to four parameters to the MAV
*/
public class msg_command_long extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_COMMAND_LONG = 76;
        public static final int MAVLINK_MSG_LENGTH = 33;
        private static final long serialVersionUID = MAVLINK_MSG_ID_COMMAND_LONG;


        /**
        * Parameter 1, as defined by MAV_CMD enum.
        */
        public float param1;
        /**
        * Parameter 2, as defined by MAV_CMD enum.
        */
        public float param2;
        /**
        * Parameter 3, as defined by MAV_CMD enum.
        */
        public float param3;
        /**
        * Parameter 4, as defined by MAV_CMD enum.
        */
        public float param4;
        /**
        * Parameter 5, as defined by MAV_CMD enum.
        */
        public float param5;
        /**
        * Parameter 6, as defined by MAV_CMD enum.
        */
        public float param6;
        /**
        * Parameter 7, as defined by MAV_CMD enum.
        */
        public float param7;
        /**
        * Command ID, as defined by MAV_CMD enum.
        */
        public short command;
        /**
        * System which should execute the command
        */
        public byte target_system;
        /**
        * Component which should execute the command, 0 for all components
        */
        public byte target_component;
        /**
        * 0: First transmission of this command. 1-255: Confirmation transmissions (e.g. for kill command)
        */
        public byte confirmation;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_COMMAND_LONG;
                packet.payload.putFloat(param1);
                packet.payload.putFloat(param2);
                packet.payload.putFloat(param3);
                packet.payload.putFloat(param4);
                packet.payload.putFloat(param5);
                packet.payload.putFloat(param6);
                packet.payload.putFloat(param7);
                packet.payload.putShort(command);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(confirmation);
                return packet;
        }

    /**
     * Decode a command_long message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            param1 = payload.getFloat();
            param2 = payload.getFloat();
            param3 = payload.getFloat();
            param4 = payload.getFloat();
            param5 = payload.getFloat();
            param6 = payload.getFloat();
            param7 = payload.getFloat();
            command = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
            confirmation = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_command_long(){
        msgid = MAVLINK_MSG_ID_COMMAND_LONG;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_command_long(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_COMMAND_LONG;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "COMMAND_LONG");
        //Log.d("MAVLINK_MSG_ID_COMMAND_LONG", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_COMMAND_LONG -"+" param1:"+param1+" param2:"+param2
+" param3:"+param3+" param4:"+param4+" param5:"+param5+" param6:"+param6+" param7:"+param
7+" command:"+command+" target_system:"+target_system+" target_component:"+target_compone
nt+" confirmation:"+confirmation+"";
    }
}
```

```java
// MESSAGE DATA16 PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Data packet, size 16
*/
public class msg_data16 extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DATA16 = 169;
        public static final int MAVLINK_MSG_LENGTH = 18;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DATA16;


        /**
        * data type
        */
        public byte type;
        /**
        * data length
        */
        public byte len;
        /**
        * raw data
        */
        public byte data[] = new byte[16];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DATA16;
                packet.payload.putByte(type);
                packet.payload.putByte(len);
                 for (int i = 0; i < data.length; i++) {
                        packet.payload.putByte(data[i]);
            }
                return packet;
        }

    /**
     * Decode a data16 message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            type = payload.getByte();
            len = payload.getByte();
             for (int i = 0; i < data.length; i++) {
                        data[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_data16(){
        msgid = MAVLINK_MSG_ID_DATA16;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_data16(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DATA16;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DATA16");
        //Log.d("MAVLINK_MSG_ID_DATA16", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DATA16 -"+" type:"+type+" len:"+len+" data:"+data+"";
    }
}
```

```java
// MESSAGE DATA32 PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Data packet, size 32
*/
public class msg_data32 extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DATA32 = 170;
        public static final int MAVLINK_MSG_LENGTH = 34;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DATA32;


        /**
        * data type
        */
        public byte type;
        /**
        * data length
        */
        public byte len;
        /**
        * raw data
        */
        public byte data[] = new byte[32];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DATA32;
                packet.payload.putByte(type);
                packet.payload.putByte(len);
                 for (int i = 0; i < data.length; i++) {
                        packet.payload.putByte(data[i]);
            }
                return packet;
        }

    /**
     * Decode a data32 message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            type = payload.getByte();
            len = payload.getByte();
             for (int i = 0; i < data.length; i++) {
                        data[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_data32(){
        msgid = MAVLINK_MSG_ID_DATA32;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_data32(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DATA32;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DATA32");
        //Log.d("MAVLINK_MSG_ID_DATA32", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DATA32 -"+" type:"+type+" len:"+len+" data:"+data+"
";
    }
}
```

```java
// MESSAGE DATA64 PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Data packet, size 64
*/
public class msg_data64 extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DATA64 = 171;
        public static final int MAVLINK_MSG_LENGTH = 66;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DATA64;


        /**
        * data type
        */
        public byte type;
        /**
        * data length
        */
        public byte len;
        /**
        * raw data
        */
        public byte data[] = new byte[64];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DATA64;
                packet.payload.putByte(type);
                packet.payload.putByte(len);
                 for (int i = 0; i < data.length; i++) {
                        packet.payload.putByte(data[i]);
            }
                return packet;
        }

    /**
     * Decode a data64 message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            type = payload.getByte();
            len = payload.getByte();
             for (int i = 0; i < data.length; i++) {
                        data[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_data64(){
        msgid = MAVLINK_MSG_ID_DATA64;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_data64(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DATA64;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DATA64");
        //Log.d("MAVLINK_MSG_ID_DATA64", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DATA64 -"+" type:"+type+" len:"+len+" data:"+data+"
";
    }
}
```

```java
// MESSAGE DATA96 PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Data packet, size 96
*/
public class msg_data96 extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DATA96 = 172;
        public static final int MAVLINK_MSG_LENGTH = 98;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DATA96;


        /**
        * data type
        */
        public byte type;
        /**
        * data length
        */
        public byte len;
        /**
        * raw data
        */
        public byte data[] = new byte[96];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DATA96;
                packet.payload.putByte(type);
                packet.payload.putByte(len);
                 for (int i = 0; i < data.length; i++) {
                        packet.payload.putByte(data[i]);
            }
                return packet;
        }

    /**
     * Decode a data96 message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            type = payload.getByte();
            len = payload.getByte();
             for (int i = 0; i < data.length; i++) {
                        data[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_data96(){
        msgid = MAVLINK_MSG_ID_DATA96;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_data96(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DATA96;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DATA96");
        //Log.d("MAVLINK_MSG_ID_DATA96", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DATA96 -"+" type:"+type+" len:"+len+" data:"+data+"
";

    }
}
```

```java
// MESSAGE DATA_STREAM PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
*
*/
public class msg_data_stream extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DATA_STREAM = 67;
        public static final int MAVLINK_MSG_LENGTH = 4;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DATA_STREAM;


        /**
        * The requested interval between two messages of this type
        */
        public short message_rate;
        /**
        * The ID of the requested data stream
        */
        public byte stream_id;
        /**
        * 1 stream is enabled, 0 stream is stopped.
        */
        public byte on_off;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DATA_STREAM;
                packet.payload.putShort(message_rate);
                packet.payload.putByte(stream_id);
                packet.payload.putByte(on_off);
                return packet;
        }

    /**
     * Decode a data_stream message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            message_rate = payload.getShort();
            stream_id = payload.getByte();
            on_off = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_data_stream(){
        msgid = MAVLINK_MSG_ID_DATA_STREAM;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_data_stream(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DATA_STREAM;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DATA_STREAM");
        //Log.d("MAVLINK_MSG_ID_DATA_STREAM", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DATA_STREAM -"+" message_rate:"+message_rate+" stre
am_id:"+stream_id+" on_off:"+on_off+"";
    }
}
```

```java
// MESSAGE DEBUG_VECT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
*
*/
public class msg_debug_vect extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DEBUG_VECT = 250;
        public static final int MAVLINK_MSG_LENGTH = 30;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DEBUG_VECT;


        /**
        * Timestamp
        */
        public long time_usec;
        /**
        * x
        */
        public float x;
        /**
        * y
        */
        public float y;
        /**
        * z
        */
        public float z;
        /**
        * Name
        */
        public byte name[] = new byte[10];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DEBUG_VECT;
                packet.payload.putLong(time_usec);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                 for (int i = 0; i < name.length; i++) {
                        packet.payload.putByte(name[i]);
            }
                return packet;
        }

    /**
     * Decode a debug_vect message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
             for (int i = 0; i < name.length; i++) {
                        name[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_debug_vect(){
        msgid = MAVLINK_MSG_ID_DEBUG_VECT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_debug_vect(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DEBUG_VECT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DEBUG_VECT");
        //Log.d("MAVLINK_MSG_ID_DEBUG_VECT", toString());
    }

         /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setName(String str) {
      int len = Math.min(str.length(), 10);
      for (int i=0; i<len; i++) {
        name[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<10; i++) {                      // padding for the rest of
the buffer
        name[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getName() {
                String result = "";
                for (int i = 0; i < 10; i++) {
                        if (name[i] != 0)
                                result = result + (char) name[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DEBUG_VECT -"+" time_usec:"+time_usec+" x:"+x+" y:"
+y+" z:"+z+" name:"+name+"";
    }
}
```

```java
// MESSAGE DIGICAM_CONFIGURE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Configure on-board Camera Control System.
*/
public class msg_digicam_configure extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DIGICAM_CONFIGURE = 154;
        public static final int MAVLINK_MSG_LENGTH = 15;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DIGICAM_CONFIGURE;


        /**
        * Correspondent value to given extra_param
        */
        public float extra_value;
        /**
        * Divisor number //e.g. 1000 means 1/1000 (0 means ignore)
        */
        public short shutter_speed;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * Mode enumeration from 1 to N //P, TV, AV, M, Etc (0 means ignore)
        */
        public byte mode;
        /**
        * F stop number x 10 //e.g. 28 means 2.8 (0 means ignore)
        */
        public byte aperture;
        /**
        * ISO enumeration from 1 to N //e.g. 80, 100, 200, Etc (0 means ignore)
        */
        public byte iso;
        /**
        * Exposure type enumeration from 1 to N (0 means ignore)
        */
        public byte exposure_type;
        /**
        * Command Identity (incremental loop: 0 to 255)//A command sent multiple times will be executed or pooled just once
        */
        public byte command_id;
        /**
        * Main engine cut-off time before camera trigger in seconds/10 (0 means no cut-off)
        */
        public byte engine_cut_off;
        /**
        * Extra parameters enumeration (0 means ignore)
        */
        public byte extra_param;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DIGICAM_CONFIGURE;
                packet.payload.putFloat(extra_value);
                packet.payload.putShort(shutter_speed);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(mode);
                packet.payload.putByte(aperture);
                packet.payload.putByte(iso);
                packet.payload.putByte(exposure_type);
                packet.payload.putByte(command_id);
                packet.payload.putByte(engine_cut_off);
                packet.payload.putByte(extra_param);
                return packet;
        }

    /**
     * Decode a digicam_configure message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            extra_value = payload.getFloat();
            shutter_speed = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
            mode = payload.getByte();
            aperture = payload.getByte();
            iso = payload.getByte();
            exposure_type = payload.getByte();
            command_id = payload.getByte();
            engine_cut_off = payload.getByte();
            extra_param = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_digicam_configure(){
        msgid = MAVLINK_MSG_ID_DIGICAM_CONFIGURE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_digicam_configure(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DIGICAM_CONFIGURE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DIGICAM_CONFIGURE");
        //Log.d("MAVLINK_MSG_ID_DIGICAM_CONFIGURE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DIGICAM_CONFIGURE -"+" extra_value:"+extra_value+" shutter_speed:"+shutter_speed+" target_system:"+target_system+" target_component:"+target_component+" mode:"+mode+" aperture:"+aperture+" iso:"+iso+" exposure_type:"+exposure_type+" command_id:"+command_id+" engine_cut_off:"+engine_cut_off+" extra_param:"+extra_param+"";
    }
}
```

```java
// MESSAGE DIGICAM_CONTROL PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Control on-board Camera Control System to take shots.
*/
public class msg_digicam_control extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DIGICAM_CONTROL = 155;
        public static final int MAVLINK_MSG_LENGTH = 13;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DIGICAM_CONTROL;


        /**
        * Correspondent value to given extra_param
        */
        public float extra_value;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * 0: stop, 1: start or keep it up //Session control e.g. show/hide lens
        */
        public byte session;
        /**
        * 1 to N //Zoom’s absolute position (0 means ignore)
        */
        public byte zoom_pos;
        /**
        * -100 to 100 //Zooming step value to offset zoom from the current position
        */
        public byte zoom_step;
        /**
        * 0: unlock focus or keep unlocked, 1: lock focus or keep locked, 3: re-lock focus
        */
        public byte focus_lock;
        /**
        * 0: ignore, 1: shot or start filming
        */
        public byte shot;
        /**
        * Command Identity (incremental loop: 0 to 255)//A command sent multiple times will be executed or pooled just once
        */
        public byte command_id;
        /**
        * Extra parameters enumeration (0 means ignore)
        */
        public byte extra_param;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DIGICAM_CONTROL;
                packet.payload.putFloat(extra_value);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(session);
                packet.payload.putByte(zoom_pos);
                packet.payload.putByte(zoom_step);
                packet.payload.putByte(focus_lock);
                packet.payload.putByte(shot);
                packet.payload.putByte(command_id);
                packet.payload.putByte(extra_param);
                return packet;
        }

    /**
     * Decode a digicam_control message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            extra_value = payload.getFloat();
            target_system = payload.getByte();
            target_component = payload.getByte();
            session = payload.getByte();
            zoom_pos = payload.getByte();
            zoom_step = payload.getByte();
            focus_lock = payload.getByte();
            shot = payload.getByte();
            command_id = payload.getByte();
            extra_param = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_digicam_control(){
        msgid = MAVLINK_MSG_ID_DIGICAM_CONTROL;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_digicam_control(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DIGICAM_CONTROL;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DIGICAM_CONTROL");
        //Log.d("MAVLINK_MSG_ID_DIGICAM_CONTROL", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DIGICAM_CONTROL -"+" extra_value:"+extra_value+" target_system:"+target_system+" target_component:"+target_component+" session:"+session+" zoom_pos:"+zoom_pos+" zoom_step:"+zoom_step+" focus_lock:"+focus_lock+" shot:"+shot+" comm
and_id:"+command_id+" extra_param:"+extra_param+"";
    }
}
```

```java
// MESSAGE FENCE_FETCH_POINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Request a current fence point from MAV
*/
public class msg_fence_fetch_point extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_FENCE_FETCH_POINT = 161;
        public static final int MAVLINK_MSG_LENGTH = 3;
        private static final long serialVersionUID = MAVLINK_MSG_ID_FENCE_FETCH_POINT;


        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * point index (first point is 1, 0 is for return point)
        */
        public byte idx;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_FENCE_FETCH_POINT;
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(idx);
                return packet;
        }

    /**
     * Decode a fence_fetch_point message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            target_system = payload.getByte();
            target_component = payload.getByte();
            idx = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_fence_fetch_point(){
        msgid = MAVLINK_MSG_ID_FENCE_FETCH_POINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_fence_fetch_point(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_FENCE_FETCH_POINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "FENCE_FETCH_POINT");
        //Log.d("MAVLINK_MSG_ID_FENCE_FETCH_POINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_FENCE_FETCH_POINT -"+" target_system:"+target_system+" target_component:"+target_component+" idx:"+idx+"";
    }
}
```

```java
// MESSAGE FENCE_POINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* A fence point. Used to set a point when from
              GCS -> MAV. Also used to return a point from MAV -> GCS
*/
public class msg_fence_point extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_FENCE_POINT = 160;
        public static final int MAVLINK_MSG_LENGTH = 12;
        private static final long serialVersionUID = MAVLINK_MSG_ID_FENCE_POINT;


        /**
        * Latitude of point
        */
        public float lat;
        /**
        * Longitude of point
        */
        public float lng;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * point index (first point is 1, 0 is for return point)
        */
        public byte idx;
        /**
        * total number of points (for sanity checking)
        */
        public byte count;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_FENCE_POINT;
                packet.payload.putFloat(lat);
                packet.payload.putFloat(lng);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(idx);
                packet.payload.putByte(count);
                return packet;
        }

    /**
     * Decode a fence_point message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            lat = payload.getFloat();
            lng = payload.getFloat();
            target_system = payload.getByte();
            target_component = payload.getByte();
            idx = payload.getByte();
            count = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_fence_point(){
        msgid = MAVLINK_MSG_ID_FENCE_POINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_fence_point(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_FENCE_POINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "FENCE_POINT");
        //Log.d("MAVLINK_MSG_ID_FENCE_POINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_FENCE_POINT -"+" lat:"+lat+" lng:"+lng+" target_sys
tem:"+target_system+" target_component:"+target_component+" idx:"+idx+" count:"+count+"";
    }
}
```

```java
// MESSAGE FENCE_STATUS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Status of geo-fencing. Sent in extended
            status stream when fencing enabled
*/
public class msg_fence_status extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_FENCE_STATUS = 162;
        public static final int MAVLINK_MSG_LENGTH = 8;
        private static final long serialVersionUID = MAVLINK_MSG_ID_FENCE_STATUS;


        /**
        * time of last breach in milliseconds since boot
        */
        public int breach_time;
        /**
        * number of fence breaches
        */
        public short breach_count;
        /**
        * 0 if currently inside fence, 1 if outside
        */
        public byte breach_status;
        /**
        * last breach type (see FENCE_BREACH_* enum)
        */
        public byte breach_type;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_FENCE_STATUS;
                packet.payload.putInt(breach_time);
                packet.payload.putShort(breach_count);
                packet.payload.putByte(breach_status);
                packet.payload.putByte(breach_type);
                return packet;
        }

    /**
     * Decode a fence_status message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            breach_time = payload.getInt();
            breach_count = payload.getShort();
            breach_status = payload.getByte();
            breach_type = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_fence_status(){
        msgid = MAVLINK_MSG_ID_FENCE_STATUS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_fence_status(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_FENCE_STATUS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "FENCE_STATUS");
        //Log.d("MAVLINK_MSG_ID_FENCE_STATUS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_FENCE_STATUS -"+" breach_time:"+breach_time+" breac
h_count:"+breach_count+" breach_status:"+breach_status+" breach_type:"+breach_type+"";
    }
}
```

```java
// MESSAGE FILE_TRANSFER_DIR_LIST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Get directory listing
*/
public class msg_file_transfer_dir_list extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST = 111;
        public static final int MAVLINK_MSG_LENGTH = 249;
        private static final long serialVersionUID = MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST;


        /**
        * Unique transfer ID
        */
        public long transfer_uid;
        /**
        * Directory path to list
        */
        public byte dir_path[] = new byte[240];
        /**
        * RESERVED
        */
        public byte flags;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST;
                packet.payload.putLong(transfer_uid);
                 for (int i = 0; i < dir_path.length; i++) {
                        packet.payload.putByte(dir_path[i]);
            }
                packet.payload.putByte(flags);
                return packet;
        }

    /**
     * Decode a file_transfer_dir_list message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            transfer_uid = payload.getLong();
             for (int i = 0; i < dir_path.length; i++) {
                        dir_path[i] = payload.getByte();
                }
            flags = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_file_transfer_dir_list(){
        msgid = MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_file_transfer_dir_list(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "FILE_TRANSFER_DIR_LIST");
        //Log.d("MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST", toString());
    }

   /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setDir_Path(String str) {
      int len = Math.min(str.length(), 240);
      for (int i=0; i<len; i++) {
        dir_path[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<240; i++) {                     // padding for the rest of the buffer
        dir_path[i] = 0;
      }
    }

    /**
        * Gets the message, formated as a string
        */
        public String getDir_Path() {
                String result = "";
                for (int i = 0; i < 240; i++) {
                        if (dir_path[i] != 0)
                                result = result + (char) dir_path[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST -"+" transfer_uid:"+transfer_uid+" dir_path:"+dir_path+" flags:"+flags+"";
    }
}
```

```java
// MESSAGE FILE_TRANSFER_RES PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* File transfer result
*/
public class msg_file_transfer_res extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_FILE_TRANSFER_RES = 112;
        public static final int MAVLINK_MSG_LENGTH = 9;
        private static final long serialVersionUID = MAVLINK_MSG_ID_FILE_TRANSFER_RES;


        /**
        * Unique transfer ID
        */
        public long transfer_uid;
        /**
        * 0: OK, 1: not permitted, 2: bad path / file name, 3: no space left on device
        */
        public byte result;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_RES;
                packet.payload.putLong(transfer_uid);
                packet.payload.putByte(result);
                return packet;
        }

    /**
     * Decode a file_transfer_res message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            transfer_uid = payload.getLong();
            result = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_file_transfer_res(){
        msgid = MAVLINK_MSG_ID_FILE_TRANSFER_RES;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_file_transfer_res(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_RES;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "FILE_TRANSFER_RES");
        //Log.d("MAVLINK_MSG_ID_FILE_TRANSFER_RES", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_FILE_TRANSFER_RES -"+" transfer_uid:"+transfer_uid+" result:"+result+"";
    }
}
```

```java
// MESSAGE FILE_TRANSFER_START PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Begin file transfer
*/
public class msg_file_transfer_start extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_FILE_TRANSFER_START = 110;
        public static final int MAVLINK_MSG_LENGTH = 254;
        private static final long serialVersionUID = MAVLINK_MSG_ID_FILE_TRANSFER_START;


        /**
        * Unique transfer ID
        */
        public long transfer_uid;
        /**
        * File size in bytes
        */
        public int file_size;
        /**
        * Destination path
        */
        public byte dest_path[] = new byte[240];
        /**
        * Transfer direction: 0: from requester, 1: to requester
        */
        public byte direction;
        /**
        * RESERVED
        */
        public byte flags;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_START;
                packet.payload.putLong(transfer_uid);
                packet.payload.putInt(file_size);
                 for (int i = 0; i < dest_path.length; i++) {
                        packet.payload.putByte(dest_path[i]);
            }
                packet.payload.putByte(direction);
                packet.payload.putByte(flags);
                return packet;
        }

    /**
     * Decode a file_transfer_start message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            transfer_uid = payload.getLong();
            file_size = payload.getInt();
             for (int i = 0; i < dest_path.length; i++) {
                        dest_path[i] = payload.getByte();
                }
            direction = payload.getByte();
            flags = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_file_transfer_start(){
        msgid = MAVLINK_MSG_ID_FILE_TRANSFER_START;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_file_transfer_start(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_START;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "FILE_TRANSFER_START");
        //Log.d("MAVLINK_MSG_ID_FILE_TRANSFER_START", toString());
    }

     /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setDest_Path(String str) {
      int len = Math.min(str.length(), 240);
      for (int i=0; i<len; i++) {
        dest_path[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<240; i++) {                     // padding for the rest of the buffer
        dest_path[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getDest_Path() {
                String result = "";
                for (int i = 0; i < 240; i++) {
                        if (dest_path[i] != 0)
                                result = result + (char) dest_path[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_FILE_TRANSFER_START -"+" transfer_uid:"+transfer_uid+" file_size:"+file_size+" dest_path:"+dest_path+" direction:"+direction+" flags:"+flags+"";
    }
}
```

```java
// MESSAGE GLOBAL_POSITION_INT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The filtered global position (e.g. fused GPS and accelerometers). The position is in GPS-frame (right-handed, Z-up). It
                is designed as scaled integer message since the resolution of float is not sufficient.
*/
public class msg_global_position_int extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_GLOBAL_POSITION_INT = 33;
        public static final int MAVLINK_MSG_LENGTH = 28;
        private static final long serialVersionUID = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * Latitude, expressed as * 1E7
        */
        public int lat;
        /**
        * Longitude, expressed as * 1E7
        */
        public int lon;
        /**
        * Altitude in meters, expressed as * 1000 (millimeters), above MSL
        */
        public int alt;
        /**
        * Altitude above ground in meters, expressed as * 1000 (millimeters)
        */
        public int relative_alt;
        /**
        * Ground X Speed (Latitude), expressed as m/s * 100
        */
        public short vx;
        /**
        * Ground Y Speed (Longitude), expressed as m/s * 100
        */
        public short vy;
        /**
        * Ground Z Speed (Altitude), expressed as m/s * 100
        */
        public short vz;
        /**
        * Compass heading in degrees * 100, 0.0..359.99 degrees. If unknown, set to: 65535
        */
        public short hdg;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putInt(lat);
                packet.payload.putInt(lon);
                packet.payload.putInt(alt);
                packet.payload.putInt(relative_alt);
                packet.payload.putShort(vx);
                packet.payload.putShort(vy);
                packet.payload.putShort(vz);
                packet.payload.putShort(hdg);
                return packet;
        }

    /**
     * Decode a global_position_int message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            lat = payload.getInt();
            lon = payload.getInt();
            alt = payload.getInt();
            relative_alt = payload.getInt();
            vx = payload.getShort();
            vy = payload.getShort();
            vz = payload.getShort();
            hdg = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_global_position_int(){
        msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_global_position_int(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "GLOBAL_POSITION_INT");
        //Log.d("MAVLINK_MSG_ID_GLOBAL_POSITION_INT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_GLOBAL_POSITION_INT -"+" time_boot_ms:"+time_boot_ms+" lat:"+lat+" lon:"+lon+" alt:"+alt+" relative_alt:"+relative_alt+" vx:"+vx+" vy:"+vy+" vz:"+vz+" hdg:"+hdg+"";
    }
}
```

```java
// MESSAGE GLOBAL_POSITION_SETPOINT_INT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Transmit the current local setpoint of the controller to other MAVs (collision avoidance) and to the GCS.
*/
public class msg_global_position_setpoint_int extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT = 52;
        public static final int MAVLINK_MSG_LENGTH = 15;
        private static final long serialVersionUID = MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT;


        /**
        * WGS84 Latitude position in degrees * 1E7
        */
        public int latitude;
        /**
        * WGS84 Longitude position in degrees * 1E7
        */
        public int longitude;
        /**
        * WGS84 Altitude in meters * 1000 (positive for up)
        */
        public int altitude;
        /**
        * Desired yaw angle in degrees * 100
        */
        public short yaw;
        /**
        * Coordinate frame - valid values are only MAV_FRAME_GLOBAL or MAV_FRAME_GLOBAL_RELATIVE_ALT
        */
        public byte coordinate_frame;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT;
                packet.payload.putInt(latitude);
                packet.payload.putInt(longitude);
                packet.payload.putInt(altitude);
                packet.payload.putShort(yaw);
                packet.payload.putByte(coordinate_frame);
                return packet;
        }

    /**
     * Decode a global_position_setpoint_int message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            latitude = payload.getInt();
            longitude = payload.getInt();
            altitude = payload.getInt();
            yaw = payload.getShort();
            coordinate_frame = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_global_position_setpoint_int(){
        msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_global_position_setpoint_int(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "GLOBAL_POSITION_SETPOINT_INT");
        //Log.d("MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT -"+" latitude:"+latitude+" longitude:"+longitude+" altitude:"+altitude+" yaw:"+yaw+" coordinate_frame:"+coordinate_frame+"";
    }
}
```

```java
// MESSAGE GLOBAL_VISION_POSITION_ESTIMATE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
*
*/
public class msg_global_vision_position_estimate extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE = 101;
        public static final int MAVLINK_MSG_LENGTH = 32;
        private static final long serialVersionUID = MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE;


        /**
        * Timestamp (microseconds, synced to UNIX time or since system boot)
        */
        public long usec;
        /**
        * Global X position
        */
        public float x;
        /**
        * Global Y position
        */
        public float y;
        /**
        * Global Z position
        */
        public float z;
        /**
        * Roll angle in rad
        */
        public float roll;
        /**
        * Pitch angle in rad
        */
        public float pitch;
        /**
        * Yaw angle in rad
        */
        public float yaw;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE;
                packet.payload.putLong(usec);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                return packet;
        }

    /**
     * Decode a global_vision_position_estimate message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            usec = payload.getLong();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_global_vision_position_estimate(){
        msgid = MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_global_vision_position_estimate(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "GLOBAL_VISION_POSITION_ESTIMATE");
        //Log.d("MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE -"+" usec:"+usec+" x:"+x+" y:"+y+" z:"+z+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+"";
    }
}
```

```java
// MESSAGE GPS_GLOBAL_ORIGIN PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Once the MAV sets a new GPS-Local correspondence, this message announces the origin (0,0,0) position
*/
public class msg_gps_global_origin extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN = 49;
        public static final int MAVLINK_MSG_LENGTH = 12;
        private static final long serialVersionUID = MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN;


        /**
        * Latitude (WGS84), expressed as * 1E7
        */
        public int latitude;
        /**
        * Longitude (WGS84), expressed as * 1E7
        */
        public int longitude;
        /**
        * Altitude(WGS84), expressed as * 1000
        */
        public int altitude;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN;
                packet.payload.putInt(latitude);
                packet.payload.putInt(longitude);
                packet.payload.putInt(altitude);
                return packet;
        }

    /**
     * Decode a gps_global_origin message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            latitude = payload.getInt();
            longitude = payload.getInt();
            altitude = payload.getInt();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_gps_global_origin(){
        msgid = MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_gps_global_origin(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "GPS_GLOBAL_ORIGIN");
        //Log.d("MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN -"+" latitude:"+latitude+" longitude:"+longitude+" altitude:"+altitude+"";
    }
}
```

```java
// MESSAGE GPS_RAW_INT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The global position, as returned by the Global Positioning System (GPS). This is
                NOT the global position estimate of the sytem, but rather a RAW sensor value. See message GLOBAL_POSITION for the global position estimate. Coordinate frame is right-handed, Z-axis up (GPS frame).
*/
public class msg_gps_raw_int extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_GPS_RAW_INT = 24;
        public static final int MAVLINK_MSG_LENGTH = 30;
        private static final long serialVersionUID = MAVLINK_MSG_ID_GPS_RAW_INT;


        /**
        * Timestamp (microseconds since UNIX epoch or microseconds since system boot)
        */
        public long time_usec;
        /**
        * Latitude in 1E7 degrees
        */
        public int lat;
        /**
        * Longitude in 1E7 degrees
        */
        public int lon;
        /**
        * Altitude in 1E3 meters (millimeters) above MSL
        */
        public int alt;
        /**
        * GPS HDOP horizontal dilution of position in cm (m*100). If unknown, set to: 65535
        */
        public short eph;
        /**
        * GPS VDOP horizontal dilution of position in cm (m*100). If unknown, set to: 65535
        */
        public short epv;
        /**
        * GPS ground speed (m/s * 100). If unknown, set to: 65535
        */
        public short vel;
        /**
        * Course over ground (NOT heading, but direction of movement) in degrees * 100, 0.0..359.99 degrees. If unknown, set to: 65535
        */
        public short cog;
        /**
        * 0-1: no fix, 2: 2D fix, 3: 3D fix. Some applications will not use the value of this field unless it is at least two, so always correctly fill in the fix.
        */
        public byte fix_type;
        /**
        * Number of satellites visible. If unknown, set to 255
        */
        public byte satellites_visible;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_GPS_RAW_INT;
                packet.payload.putLong(time_usec);
                packet.payload.putInt(lat);
                packet.payload.putInt(lon);
                packet.payload.putInt(alt);
                packet.payload.putShort(eph);
                packet.payload.putShort(epv);
                packet.payload.putShort(vel);
                packet.payload.putShort(cog);
                packet.payload.putByte(fix_type);
                packet.payload.putByte(satellites_visible);
                return packet;
        }

    /**
     * Decode a gps_raw_int message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            lat = payload.getInt();
            lon = payload.getInt();
            alt = payload.getInt();
            eph = payload.getShort();
            epv = payload.getShort();
            vel = payload.getShort();
            cog = payload.getShort();
            fix_type = payload.getByte();
            satellites_visible = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_gps_raw_int(){
        msgid = MAVLINK_MSG_ID_GPS_RAW_INT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_gps_raw_int(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_GPS_RAW_INT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "GPS_RAW_INT");
        //Log.d("MAVLINK_MSG_ID_GPS_RAW_INT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_GPS_RAW_INT -"+" time_usec:"+time_usec+" lat:"+lat+" lon:"+lon+" alt:"+alt+" eph:"+eph+" epv:"+epv+" vel:"+vel+" cog:"+cog+" fix_type:"+fix_type+" satellites_visible:"+satellites_visible+"";
    }
}
```

```java
// MESSAGE HEARTBEAT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The heartbeat message shows that a system is present and responding. The type of the MAV and Autopilot hardware allow the receiving system to treat further messages from this system appropriate (e.g. by laying out the user interface based on the autopilot).
*/
public class msg_heartbeat extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_HEARTBEAT = 0;
        public static final int MAVLINK_MSG_LENGTH = 9;
        private static final long serialVersionUID = MAVLINK_MSG_ID_HEARTBEAT;


        /**
        * A bitfield for use for autopilot-specific flags.
        */
        public int custom_mode;
        /**
        * Type of the MAV (quadrotor, helicopter, etc., up to 15 types, defined in MAV_TYPE ENUM)
        */
        public byte type;
        /**
        * Autopilot type / class. defined in MAV_AUTOPILOT ENUM
        */
        public byte autopilot;
        /**
        * System mode bitfield, see MAV_MODE_FLAGS ENUM in mavlink/include/mavlink_types.h
        */
        public byte base_mode;
        /**
        * System status flag, see MAV_STATE ENUM
        */
        public byte system_status;
        /**
        * MAVLink version, not writable by user, gets added by protocol because of magic data type: uint8_t_mavlink_version
        */
        public byte mavlink_version;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_HEARTBEAT;
                packet.payload.putInt(custom_mode);
                packet.payload.putByte(type);
                packet.payload.putByte(autopilot);
                packet.payload.putByte(base_mode);
                packet.payload.putByte(system_status);
                packet.payload.putByte(mavlink_version);
                return packet;
        }

    /**
     * Decode a heartbeat message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            custom_mode = payload.getInt();
            type = payload.getByte();
            autopilot = payload.getByte();
            base_mode = payload.getByte();
            system_status = payload.getByte();
            mavlink_version = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_heartbeat(){
        msgid = MAVLINK_MSG_ID_HEARTBEAT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_heartbeat(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_HEARTBEAT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "HEARTBEAT");
        //Log.d("MAVLINK_MSG_ID_HEARTBEAT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_HEARTBEAT -"+" custom_mode:"+custom_mode+" type:"+type+" autopilot:"+autopilot+" base_mode:"+base_mode+" system_status:"+system_status+" mavlink_version:"+mavlink_version+"";
    }
}
```

```java
// MESSAGE HIGHRES_IMU PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The IMU readings in SI units in NED body frame
*/
public class msg_highres_imu extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_HIGHRES_IMU = 105;
        public static final int MAVLINK_MSG_LENGTH = 62;
        private static final long serialVersionUID = MAVLINK_MSG_ID_HIGHRES_IMU;


        /**
        * Timestamp (microseconds, synced to UNIX time or since system boot)
        */
        public long time_usec;
        /**
        * X acceleration (m/s^2)
        */
        public float xacc;
        /**
        * Y acceleration (m/s^2)
        */
        public float yacc;
        /**
        * Z acceleration (m/s^2)
        */
        public float zacc;
        /**
        * Angular speed around X axis (rad / sec)
        */
        public float xgyro;
        /**
        * Angular speed around Y axis (rad / sec)
        */
        public float ygyro;
        /**
        * Angular speed around Z axis (rad / sec)
        */
        public float zgyro;
        /**
        * X Magnetic field (Gauss)
        */
        public float xmag;
        /**
        * Y Magnetic field (Gauss)
        */
        public float ymag;
        /**
        * Z Magnetic field (Gauss)
        */
        public float zmag;
        /**
        * Absolute pressure in millibar
        */
        public float abs_pressure;
        /**
        * Differential pressure in millibar
        */
        public float diff_pressure;
        /**
        * Altitude calculated from pressure
        */
        public float pressure_alt;
        /**
        * Temperature in degrees celsius
        */
        public float temperature;
        /**
        * Bitmask for fields that have updated since last message, bit 0 = xacc, bit 12: temperature
        */
        public short fields_updated;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_HIGHRES_IMU;
                packet.payload.putLong(time_usec);
                packet.payload.putFloat(xacc);
                packet.payload.putFloat(yacc);
                packet.payload.putFloat(zacc);
                packet.payload.putFloat(xgyro);
                packet.payload.putFloat(ygyro);
                packet.payload.putFloat(zgyro);
                packet.payload.putFloat(xmag);
                packet.payload.putFloat(ymag);
                packet.payload.putFloat(zmag);
                packet.payload.putFloat(abs_pressure);
                packet.payload.putFloat(diff_pressure);
                packet.payload.putFloat(pressure_alt);
                packet.payload.putFloat(temperature);
                packet.payload.putShort(fields_updated);
                return packet;
        }

    /**
     * Decode a highres_imu message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            xacc = payload.getFloat();
            yacc = payload.getFloat();
            zacc = payload.getFloat();
            xgyro = payload.getFloat();
            ygyro = payload.getFloat();
            zgyro = payload.getFloat();
            xmag = payload.getFloat();
            ymag = payload.getFloat();
            zmag = payload.getFloat();
            abs_pressure = payload.getFloat();
            diff_pressure = payload.getFloat();
            pressure_alt = payload.getFloat();
            temperature = payload.getFloat();
            fields_updated = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_highres_imu(){
        msgid = MAVLINK_MSG_ID_HIGHRES_IMU;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_highres_imu(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_HIGHRES_IMU;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "HIGHRES_IMU");
        //Log.d("MAVLINK_MSG_ID_HIGHRES_IMU", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_HIGHRES_IMU -"+" time_usec:"+time_usec+" xacc:"+xacc+" yacc:"+yacc+" zacc:"+zacc+" xgyro:"+xgyro+" ygyro:"+ygyro+" zgyro:"+zgyro+" xmag:"+xmag+" ymag:"+ymag+" zmag:"+zmag+" abs_pressure:"+abs_pressure+" diff_pressure:"+diff_pressure+" pressure_alt:"+pressure_alt+" temperature:"+temperature+" fields_updated:"+fields_updated+"";
    }
}
```

```java
// MESSAGE HIL_CONTROLS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Sent from autopilot to simulation. Hardware in the loop control outputs
*/
public class msg_hil_controls extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_HIL_CONTROLS = 91;
        public static final int MAVLINK_MSG_LENGTH = 42;
        private static final long serialVersionUID = MAVLINK_MSG_ID_HIL_CONTROLS;


        /**
        * Timestamp (microseconds since UNIX epoch or microseconds since system bo
ot)
        */
        public long time_usec;
        /**
        * Control output -1 .. 1
        */
        public float roll_ailerons;
        /**
        * Control output -1 .. 1
        */
        public float pitch_elevator;
        /**
        * Control output -1 .. 1
        */
        public float yaw_rudder;
        /**
        * Throttle 0 .. 1
        */
        public float throttle;
        /**
        * Aux 1, -1 .. 1
        */
        public float aux1;
        /**
        * Aux 2, -1 .. 1
        */
        public float aux2;
        /**
        * Aux 3, -1 .. 1
        */
        public float aux3;
        /**
        * Aux 4, -1 .. 1
        */
        public float aux4;
        /**
        * System mode (MAV_MODE)
        */
        public byte mode;
        /**
        * Navigation mode (MAV_NAV_MODE)
        */
        public byte nav_mode;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_HIL_CONTROLS;
                packet.payload.putLong(time_usec);
                packet.payload.putFloat(roll_ailerons);
                packet.payload.putFloat(pitch_elevator);
                packet.payload.putFloat(yaw_rudder);
                packet.payload.putFloat(throttle);
                packet.payload.putFloat(aux1);
                packet.payload.putFloat(aux2);
                packet.payload.putFloat(aux3);
                packet.payload.putFloat(aux4);
                packet.payload.putByte(mode);
                packet.payload.putByte(nav_mode);
                return packet;
        }

    /**
     * Decode a hil_controls message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            roll_ailerons = payload.getFloat();
            pitch_elevator = payload.getFloat();
            yaw_rudder = payload.getFloat();
            throttle = payload.getFloat();
            aux1 = payload.getFloat();
            aux2 = payload.getFloat();
            aux3 = payload.getFloat();
            aux4 = payload.getFloat();
            mode = payload.getByte();
            nav_mode = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_hil_controls(){
        msgid = MAVLINK_MSG_ID_HIL_CONTROLS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_hil_controls(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_HIL_CONTROLS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "HIL_CONTROLS");
        //Log.d("MAVLINK_MSG_ID_HIL_CONTROLS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_HIL_CONTROLS -"+" time_usec:"+time_usec+" roll_ailerons:"+roll_ailerons+" pitch_elevator:"+pitch_elevator+" yaw_rudder:"+yaw_rudder+" throttle:"+throttle+" aux1:"+aux1+" aux2:"+aux2+" aux3:"+aux3+" aux4:"+aux4+" mode:"+mode+" nav_mode:"+nav_mode+"";
    }
}
```

```java
// MESSAGE HIL_RC_INPUTS_RAW PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Sent from simulation to autopilot. The RAW values of the RC channels received. The standard PPM modulation is as follows: 1000 microseconds: 0%, 2000 microseconds: 100%. Individual receivers/transmitters might violate this specification.
*/
public class msg_hil_rc_inputs_raw extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW = 92;
        public static final int MAVLINK_MSG_LENGTH = 33;
        private static final long serialVersionUID = MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW;


        /**
        * Timestamp (microseconds since UNIX epoch or microseconds since system boot)
        */
        public long time_usec;
        /**
        * RC channel 1 value, in microseconds
        */
        public short chan1_raw;
        /**
        * RC channel 2 value, in microseconds
        */
        public short chan2_raw;
        /**
        * RC channel 3 value, in microseconds
        */
        public short chan3_raw;
        /**
        * RC channel 4 value, in microseconds
        */
        public short chan4_raw;
        /**
        * RC channel 5 value, in microseconds
        */
        public short chan5_raw;
        /**
        * RC channel 6 value, in microseconds
        */
        public short chan6_raw;
        /**
        * RC channel 7 value, in microseconds
        */
        public short chan7_raw;
        /**
        * RC channel 8 value, in microseconds
        */
        public short chan8_raw;
        /**
        * RC channel 9 value, in microseconds
        */
        public short chan9_raw;
        /**
        * RC channel 10 value, in microseconds
        */
        public short chan10_raw;
        /**
        * RC channel 11 value, in microseconds
        */
        public short chan11_raw;
        /**
        * RC channel 12 value, in microseconds
        */
        public short chan12_raw;
        /**
        * Receive signal strength indicator, 0: 0%, 255: 100%
        */
        public byte rssi;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW;
                packet.payload.putLong(time_usec);
                packet.payload.putShort(chan1_raw);
                packet.payload.putShort(chan2_raw);
                packet.payload.putShort(chan3_raw);
                packet.payload.putShort(chan4_raw);
                packet.payload.putShort(chan5_raw);
                packet.payload.putShort(chan6_raw);
                packet.payload.putShort(chan7_raw);
                packet.payload.putShort(chan8_raw);
                packet.payload.putShort(chan9_raw);
                packet.payload.putShort(chan10_raw);
                packet.payload.putShort(chan11_raw);
                packet.payload.putShort(chan12_raw);
                packet.payload.putByte(rssi);
                return packet;
        }

    /**
     * Decode a hil_rc_inputs_raw message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            chan1_raw = payload.getShort();
            chan2_raw = payload.getShort();
            chan3_raw = payload.getShort();
            chan4_raw = payload.getShort();
            chan5_raw = payload.getShort();
            chan6_raw = payload.getShort();
            chan7_raw = payload.getShort();
            chan8_raw = payload.getShort();
            chan9_raw = payload.getShort();
            chan10_raw = payload.getShort();
            chan11_raw = payload.getShort();
            chan12_raw = payload.getShort();
            rssi = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_hil_rc_inputs_raw(){
        msgid = MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_hil_rc_inputs_raw(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "HIL_RC_INPUTS_RAW");
        //Log.d("MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW -"+" time_usec:"+time_usec+" chan1_raw:"+chan1_raw+" chan2_raw:"+chan2_raw+" chan3_raw:"+chan3_raw+" chan4_raw:"+chan4_raw+" chan5_raw:"+chan5_raw+" chan6_raw:"+chan6_raw+" chan7_raw:"+chan7_raw+" chan8_raw:"+chan8_raw+" chan9_raw:"+chan9_raw+" chan10_raw:"+chan10_raw+" chan11_raw:"+chan11_raw+" chan12_raw:"+chan12_raw+" rssi:"+rssi+"";
    }
}
```

```java
// MESSAGE HIL_STATE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Sent from simulation to autopilot. This packet is useful for high throughput applications such as hardware in the loop simulations.
*/
public class msg_hil_state extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_HIL_STATE = 90;
        public static final int MAVLINK_MSG_LENGTH = 56;
        private static final long serialVersionUID = MAVLINK_MSG_ID_HIL_STATE;


        /**
        * Timestamp (microseconds since UNIX epoch or microseconds since system boot)
        */
        public long time_usec;
        /**
        * Roll angle (rad)
        */
        public float roll;
        /**
        * Pitch angle (rad)
        */
        public float pitch;
        /**
        * Yaw angle (rad)
        */
        public float yaw;
        /**
        * Roll angular speed (rad/s)
        */
        public float rollspeed;
        /**
        * Pitch angular speed (rad/s)
        */
        public float pitchspeed;
        /**
        * Yaw angular speed (rad/s)
        */
        public float yawspeed;
        /**
        * Latitude, expressed as * 1E7
        */
        public int lat;
        /**
        * Longitude, expressed as * 1E7
        */
        public int lon;
        /**
        * Altitude in meters, expressed as * 1000 (millimeters)
        */
        public int alt;
        /**
        * Ground X Speed (Latitude), expressed as m/s * 100
        */
        public short vx;
        /**
        * Ground Y Speed (Longitude), expressed as m/s * 100
        */
        public short vy;
        /**
        * Ground Z Speed (Altitude), expressed as m/s * 100
        */
        public short vz;
        /**
        * X acceleration (mg)
        */
        public short xacc;
        /**
        * Y acceleration (mg)
        */
        public short yacc;
        /**
        * Z acceleration (mg)
        */
        public short zacc;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_HIL_STATE;
                packet.payload.putLong(time_usec);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                packet.payload.putFloat(rollspeed);
                packet.payload.putFloat(pitchspeed);
                packet.payload.putFloat(yawspeed);
                packet.payload.putInt(lat);
                packet.payload.putInt(lon);
                packet.payload.putInt(alt);
                packet.payload.putShort(vx);
                packet.payload.putShort(vy);
                packet.payload.putShort(vz);
                packet.payload.putShort(xacc);
                packet.payload.putShort(yacc);
                packet.payload.putShort(zacc);
                return packet;
        }

    /**
     * Decode a hil_state message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
            rollspeed = payload.getFloat();
            pitchspeed = payload.getFloat();
            yawspeed = payload.getFloat();
            lat = payload.getInt();
            lon = payload.getInt();
            alt = payload.getInt();
            vx = payload.getShort();
            vy = payload.getShort();
            vz = payload.getShort();
            xacc = payload.getShort();
            yacc = payload.getShort();
            zacc = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_hil_state(){
        msgid = MAVLINK_MSG_ID_HIL_STATE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_hil_state(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_HIL_STATE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "HIL_STATE");
        //Log.d("MAVLINK_MSG_ID_HIL_STATE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_HIL_STATE -"+" time_usec:"+time_usec+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+" rollspeed:"+rollspeed+" pitchspeed:"+pitchspeed+" yawspeed:"+yawspeed+" lat:"+lat+" lon:"+lon+" alt:"+alt+" vx:"+vx+" vy:"+vy+" vz:"+vz+" xacc:"+xacc+" yacc:"+yacc+" zacc:"+zacc+"";
    }
}
```

```java
// MESSAGE HWSTATUS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Status of key hardware
*/
public class msg_hwstatus extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_HWSTATUS = 165;
        public static final int MAVLINK_MSG_LENGTH = 3;
        private static final long serialVersionUID = MAVLINK_MSG_ID_HWSTATUS;


        /**
        * board voltage (mV)
        */
        public short Vcc;
        /**
        * I2C error count
        */
        public byte I2Cerr;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_HWSTATUS;
                packet.payload.putShort(Vcc);
                packet.payload.putByte(I2Cerr);
                return packet;
        }

    /**
     * Decode a hwstatus message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            Vcc = payload.getShort();
            I2Cerr = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_hwstatus(){
        msgid = MAVLINK_MSG_ID_HWSTATUS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_hwstatus(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_HWSTATUS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "HWSTATUS");
        //Log.d("MAVLINK_MSG_ID_HWSTATUS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_HWSTATUS -"+" Vcc:"+Vcc+" I2Cerr:"+I2Cerr+"";
    }
}
```

```java
// MESSAGE LIMITS_STATUS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Status of AP_Limits. Sent in extended
            status stream when AP_Limits is enabled
*/
public class msg_limits_status extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_LIMITS_STATUS = 167;
        public static final int MAVLINK_MSG_LENGTH = 22;
        private static final long serialVersionUID = MAVLINK_MSG_ID_LIMITS_STATUS;


        /**
        * time of last breach in milliseconds since boot
        */
        public int last_trigger;
        /**
        * time of last recovery action in milliseconds since boot
        */
        public int last_action;
        /**
        * time of last successful recovery in milliseconds since boot
        */
        public int last_recovery;
        /**
        * time of last all-clear in milliseconds since boot
        */
        public int last_clear;
        /**
        * number of fence breaches
        */
        public short breach_count;
        /**
        * state of AP_Limits, (see enum LimitState, LIMITS_STATE)
        */
        public byte limits_state;
        /**
        * AP_Limit_Module bitfield of enabled modules, (see enum moduleid or LIMIT_MODULE)
        */
        public byte mods_enabled;
        /**
        * AP_Limit_Module bitfield of required modules, (see enum moduleid or LIMIT_MODULE)
        */
        public byte mods_required;
        /**
        * AP_Limit_Module bitfield of triggered modules, (see enum moduleid or LIMIT_MODULE)
        */
        public byte mods_triggered;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_LIMITS_STATUS;
                packet.payload.putInt(last_trigger);
                packet.payload.putInt(last_action);
                packet.payload.putInt(last_recovery);
                packet.payload.putInt(last_clear);
                packet.payload.putShort(breach_count);
                packet.payload.putByte(limits_state);
                packet.payload.putByte(mods_enabled);
                packet.payload.putByte(mods_required);
                packet.payload.putByte(mods_triggered);
                return packet;
        }

    /**
     * Decode a limits_status message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            last_trigger = payload.getInt();
            last_action = payload.getInt();
            last_recovery = payload.getInt();
            last_clear = payload.getInt();
            breach_count = payload.getShort();
            limits_state = payload.getByte();
            mods_enabled = payload.getByte();
            mods_required = payload.getByte();
            mods_triggered = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_limits_status(){
        msgid = MAVLINK_MSG_ID_LIMITS_STATUS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_limits_status(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_LIMITS_STATUS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "LIMITS_STATUS");
        //Log.d("MAVLINK_MSG_ID_LIMITS_STATUS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_LIMITS_STATUS -"+" last_trigger:"+last_trigger+" last_action:"+last_action+" last_recovery:"+last_recovery+" last_clear:"+last_clear+" breach_count:"+breach_count+" limits_state:"+limits_state+" mods_enabled:"+mods_enabled+" mods_required:"+mods_required+" mods_triggered:"+mods_triggered+"";
    }
}
```

```java
// MESSAGE LOCAL_POSITION_NED PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The filtered local position (e.g. fused computer vision and accelerometers). Coordinate frame is right-handed, Z-axis down (aeronautical frame, NED / north-east-down convention)
*/
public class msg_local_position_ned extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_LOCAL_POSITION_NED = 32;
        public static final int MAVLINK_MSG_LENGTH = 28;
        private static final long serialVersionUID = MAVLINK_MSG_ID_LOCAL_POSITION_NED;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * X Position
        */
        public float x;
        /**
        * Y Position
        */
        public float y;
        /**
        * Z Position
        */
        public float z;
        /**
        * X Speed
        */
        public float vx;
        /**
        * Y Speed
        */
        public float vy;
        /**
        * Z Speed
        */
        public float vz;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putFloat(vx);
                packet.payload.putFloat(vy);
                packet.payload.putFloat(vz);
                return packet;
        }

    /**
     * Decode a local_position_ned message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            vx = payload.getFloat();
            vy = payload.getFloat();
            vz = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_local_position_ned(){
        msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_local_position_ned(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "LOCAL_POSITION_NED");
        //Log.d("MAVLINK_MSG_ID_LOCAL_POSITION_NED", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_LOCAL_POSITION_NED -"+" time_boot_ms:"+time_boot_ms+" x:"+x+" y:"+y+" z:"+z+" vx:"+vx+" vy:"+vy+" vz:"+vz+"";
    }
}
```

```java
// MESSAGE LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The offset in X, Y, Z and yaw between the LOCAL_POSITION_NED messages of MAV X and the global coordinate frame in NED coordinates. Coordinate frame is right-handed, Z-axis down (aeronautical frame, NED / north-east-down convention)
*/
public class msg_local_position_ned_system_global_offset extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET = 89;
        public static final int MAVLINK_MSG_LENGTH = 28;
        private static final long serialVersionUID = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * X Position
        */
        public float x;
        /**
        * Y Position
        */
        public float y;
        /**
        * Z Position
        */
        public float z;
        /**
        * Roll
        */
        public float roll;
        /**
        * Pitch
        */
        public float pitch;
        /**
        * Yaw
        */
        public float yaw;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                return packet;
        }

    /**
     * Decode a local_position_ned_system_global_offset message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_local_position_ned_system_global_offset(){
        msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_local_position_ned_system_global_offset(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET");
        //Log.d("MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET -"+" time_boot_ms:"+time_boot_ms+" x:"+x+" y:"+y+" z:"+z+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+"";
    }
}
```

```java
// MESSAGE LOCAL_POSITION_SETPOINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Transmit the current local setpoint of the controller to other MAVs (collision avoidance) and to the GCS.
*/
public class msg_local_position_setpoint extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT = 51;
        public static final int MAVLINK_MSG_LENGTH = 17;
        private static final long serialVersionUID = MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT;


        /**
        * x position
        */
        public float x;
        /**
        * y position
        */
        public float y;
        /**
        * z position
        */
        public float z;
        /**
        * Desired yaw angle
        */
        public float yaw;
        /**
        * Coordinate frame - valid values are only MAV_FRAME_LOCAL_NED or MAV_FRAME_LOCAL_ENU
        */
        public byte coordinate_frame;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT;
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putFloat(yaw);
                packet.payload.putByte(coordinate_frame);
                return packet;
        }

    /**
     * Decode a local_position_setpoint message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            yaw = payload.getFloat();
            coordinate_frame = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_local_position_setpoint(){
        msgid = MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_local_position_setpoint(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "LOCAL_POSITION_SETPOINT");
        //Log.d("MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT -"+" x:"+x+" y:"+y+" z:"+z+" yaw:"+yaw+" coordinate_frame:"+coordinate_frame+"";
    }
}
```

```java
// MESSAGE MANUAL_CONTROL PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* This message provides an API for manually controlling the vehicle using standard joystick axes nomenclature, along with a joystick-like input device. Unused axes can be disabled an buttons are also transmit as boolean values of their
*/
public class msg_manual_control extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MANUAL_CONTROL = 69;
        public static final int MAVLINK_MSG_LENGTH = 11;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MANUAL_CONTROL;


        /**
        * X-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to forward(1000)-backward(-1000) movement on a joystick and the pitch of a vehicle.
        */
        public short x;
        /**
        * Y-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to left(-1000)-right(1000) movement on a joystick and the roll of a vehicle.
        */
        public short y;
        /**
        * Z-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to a separate slider movement with maximum being 1000 and minimum being -1000 on a joystick and the thrust of a vehicle.
        */
        public short z;
        /**
        * R-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to a twisting of the joystick, with counter-clockwise being 1000 and clockwise being -1000, and the yaw of a vehicle.
        */
        public short r;
        /**
        * A bitfield corresponding to the joystick buttons’ current state, 1 for pressed, 0 for released. The lowest bit corresponds to Button 1.
        */
        public short buttons;
        /**
        * The system to be controlled.
        */
        public byte target;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MANUAL_CONTROL;
                packet.payload.putShort(x);
                packet.payload.putShort(y);
                packet.payload.putShort(z);
                packet.payload.putShort(r);
                packet.payload.putShort(buttons);
                packet.payload.putByte(target);
                return packet;
        }

    /**
     * Decode a manual_control message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            x = payload.getShort();
            y = payload.getShort();
            z = payload.getShort();
            r = payload.getShort();
            buttons = payload.getShort();
            target = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_manual_control(){
        msgid = MAVLINK_MSG_ID_MANUAL_CONTROL;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_manual_control(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MANUAL_CONTROL;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MANUAL_CONTROL");
        //Log.d("MAVLINK_MSG_ID_MANUAL_CONTROL", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MANUAL_CONTROL -"+" x:"+x+" y:"+y+" z:"+z+" r:"+r+" buttons:"+buttons+" target:"+target+"";
    }
}
```

```java
// MESSAGE MANUAL_SETPOINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Setpoint in roll, pitch, yaw and thrust from the operator
*/
public class msg_manual_setpoint extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MANUAL_SETPOINT = 81;
        public static final int MAVLINK_MSG_LENGTH = 22;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MANUAL_SETPOINT;


        /**
        * Timestamp in milliseconds since system boot
        */
        public int time_boot_ms;
        /**
        * Desired roll rate in radians per second
        */
        public float roll;
        /**
        * Desired pitch rate in radians per second
        */
        public float pitch;
        /**
        * Desired yaw rate in radians per second
        */
        public float yaw;
        /**
        * Collective thrust, normalized to 0 .. 1
        */
        public float thrust;
        /**
        * Flight mode switch position, 0.. 255
        */
        public byte mode_switch;
        /**
        * Override mode switch position, 0.. 255
        */
        public byte manual_override_switch;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MANUAL_SETPOINT;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                packet.payload.putFloat(thrust);
                packet.payload.putByte(mode_switch);
                packet.payload.putByte(manual_override_switch);
                return packet;
        }

    /**
     * Decode a manual_setpoint message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
            thrust = payload.getFloat();
            mode_switch = payload.getByte();
            manual_override_switch = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_manual_setpoint(){
        msgid = MAVLINK_MSG_ID_MANUAL_SETPOINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_manual_setpoint(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MANUAL_SETPOINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MANUAL_SETPOINT");
        //Log.d("MAVLINK_MSG_ID_MANUAL_SETPOINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MANUAL_SETPOINT -"+" time_boot_ms:"+time_boot_ms+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+" thrust:"+thrust+" mode_switch:"+mode_switch+" manual_override_switch:"+manual_override_switch+"";
    }
}
```

```java
// MESSAGE MEMINFO PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* state of APM memory
*/
public class msg_meminfo extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MEMINFO = 152;
        public static final int MAVLINK_MSG_LENGTH = 4;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MEMINFO;


        /**
        * heap top
        */
        public short brkval;
        /**
        * free memory
        */
        public short freemem;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MEMINFO;
                packet.payload.putShort(brkval);
                packet.payload.putShort(freemem);
                return packet;
        }

    /**
     * Decode a meminfo message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            brkval = payload.getShort();
            freemem = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_meminfo(){
        msgid = MAVLINK_MSG_ID_MEMINFO;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_meminfo(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MEMINFO;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MEMINFO");
        //Log.d("MAVLINK_MSG_ID_MEMINFO", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MEMINFO -"+" brkval:"+brkval+" freemem:"+freemem+""
;
    }
}
```

```java
// MESSAGE MEMORY_VECT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Send raw controller memory. The use of this message is discouraged for normal packets, but a quite efficient way for testing new messages and getting experimental debug output.
*/
public class msg_memory_vect extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MEMORY_VECT = 249;
        public static final int MAVLINK_MSG_LENGTH = 36;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MEMORY_VECT;


        /**
        * Starting address of the debug variables
        */
        public short address;
        /**
        * Version code of the type variable. 0=unknown, type ignored and assumed int16_t. 1=as below
        */
        public byte ver;
        /**
        * Type code of the memory variables. for ver = 1: 0=16 x int16_t, 1=16 x uint16_t, 2=16 x Q15, 3=16 x 1Q14
        */
        public byte type;
        /**
        * Memory contents at specified address
        */
        public byte value[] = new byte[32];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MEMORY_VECT;
                packet.payload.putShort(address);
                packet.payload.putByte(ver);
                packet.payload.putByte(type);
                 for (int i = 0; i < value.length; i++) {
                        packet.payload.putByte(value[i]);
            }
                return packet;
        }

    /**
     * Decode a memory_vect message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            address = payload.getShort();
            ver = payload.getByte();
            type = payload.getByte();
             for (int i = 0; i < value.length; i++) {
                        value[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_memory_vect(){
        msgid = MAVLINK_MSG_ID_MEMORY_VECT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_memory_vect(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MEMORY_VECT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MEMORY_VECT");
        //Log.d("MAVLINK_MSG_ID_MEMORY_VECT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MEMORY_VECT -"+" address:"+address+" ver:"+ver+" type:"+type+" value:"+value+"";
    }
}
```

```java
// MESSAGE MISSION_CLEAR_ALL PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Delete all mission items at once.
*/
public class msg_mission_clear_all extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_CLEAR_ALL = 45;
        public static final int MAVLINK_MSG_LENGTH = 2;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_CLEAR_
ALL;


        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_CLEAR_ALL;
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mission_clear_all message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_clear_all(){
        msgid = MAVLINK_MSG_ID_MISSION_CLEAR_ALL;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_clear_all(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_CLEAR_ALL;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_CLEAR_ALL");
        //Log.d("MAVLINK_MSG_ID_MISSION_CLEAR_ALL", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_CLEAR_ALL -"+" target_system:"+target_syste
m+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE MISSION_COUNT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* This message is emitted as response to MISSION_REQUEST_LIST by the MAV and to initiate a write transaction. The GCS can then request the individual mission item based on the knowledge of the total number of MISSIONs.
*/
public class msg_mission_count extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_COUNT = 44;
        public static final int MAVLINK_MSG_LENGTH = 4;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_COUNT;


        /**
        * Number of mission items in the sequence
        */
        public short count;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_COUNT;
                packet.payload.putShort(count);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mission_count message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            count = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_count(){
        msgid = MAVLINK_MSG_ID_MISSION_COUNT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_count(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_COUNT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_COUNT");
        //Log.d("MAVLINK_MSG_ID_MISSION_COUNT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_COUNT -"+" count:"+count+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE MISSION_CURRENT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Message that announces the sequence number of the current active mission item. The MAV will fly towards this mission item.
*/
public class msg_mission_current extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_CURRENT = 42;
        public static final int MAVLINK_MSG_LENGTH = 2;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_CURRENT;


        /**
        * Sequence
        */
        public short seq;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_CURRENT;
                packet.payload.putShort(seq);
                return packet;
        }

    /**
     * Decode a mission_current message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            seq = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_current(){
        msgid = MAVLINK_MSG_ID_MISSION_CURRENT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_current(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_CURRENT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_CURRENT");
        //Log.d("MAVLINK_MSG_ID_MISSION_CURRENT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_CURRENT -"+" seq:"+seq+"";
    }
}
```

```java
// MESSAGE MISSION_ITEM PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Message encoding a mission item. This message is emitted to announce
                the presence of a mission item and to set a mission item on the system. The mission item can be either in x, y, z meters (type: LOCAL) or x:lat, y:lon, z:altitude. Local frame is Z-down, right handed (NED), global frame is Z-up, right handed (ENU). See also http://qgroundcontrol.org/mavlink/waypoint_protocol.
*/
public class msg_mission_item extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_ITEM = 39;
        public static final int MAVLINK_MSG_LENGTH = 37;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_ITEM;


        /**
        * PARAM1 / For NAV command MISSIONs: Radius in which the MISSION is accepted as reached, in meters
        */
        public float param1;
        /**
        * PARAM2 / For NAV command MISSIONs: Time that the MAV should stay inside the PARAM1 radius before advancing, in milliseconds
        */
        public float param2;
        /**
        * PARAM3 / For LOITER command MISSIONs: Orbit to circle around the MISSION, in meters. If positive the orbit direction should be clockwise, if negative the orbit direction should be counter-clockwise.
        */
        public float param3;
        /**
        * PARAM4 / For NAV and LOITER command MISSIONs: Yaw orientation in degrees, [0..360] 0 = NORTH
        */
        public float param4;
        /**
        * PARAM5 / local: x position, global: latitude
        */
        public float x;
        /**
        * PARAM6 / y position: global: longitude
        */
        public float y;
        /**
        * PARAM7 / z position: global: altitude
        */
        public float z;
        /**
        * Sequence
        */
        public short seq;
        /**
        * The scheduled action for the MISSION. see MAV_CMD in common.xml MAVLink specs
        */
        public short command;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * The coordinate system of the MISSION. see MAV_FRAME in mavlink_types.h
        */
        public byte frame;
        /**
        * false:0, true:1
        */
        public byte current;
        /**
        * autocontinue to next wp
        */
        public byte autocontinue;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_ITEM;
                packet.payload.putFloat(param1);
                packet.payload.putFloat(param2);
                packet.payload.putFloat(param3);
                packet.payload.putFloat(param4);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putShort(seq);
                packet.payload.putShort(command);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(frame);
                packet.payload.putByte(current);
                packet.payload.putByte(autocontinue);
                return packet;
        }

    /**
     * Decode a mission_item message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            param1 = payload.getFloat();
            param2 = payload.getFloat();
            param3 = payload.getFloat();
            param4 = payload.getFloat();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            seq = payload.getShort();
            command = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
            frame = payload.getByte();
            current = payload.getByte();
            autocontinue = payload.getByte();
    }
```

```java

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_item(){
        msgid = MAVLINK_MSG_ID_MISSION_ITEM;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_item(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_ITEM;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_ITEM");
        //Log.d("MAVLINK_MSG_ID_MISSION_ITEM", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_ITEM -"+" param1:"+param1+" param2:"+param2
+" param3:"+param3+" param4:"+param4+" x:"+x+" y:"+y+" z:"+z+" seq:"+seq+" command:"+comm
and+" target_system:"+target_system+" target_component:"+target_component+" frame:"+frame
+" current:"+current+" autocontinue:"+autocontinue+"";
    }
}
```

```
// MESSAGE MISSION_ITEM_REACHED PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* A certain mission item has been reached. The system will either hold this position (or circle on the orbit) or (if the autocontinue on the WP was set) continue to the next MISSION.
*/
public class msg_mission_item_reached extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_ITEM_REACHED = 46;
        public static final int MAVLINK_MSG_LENGTH = 2;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;


        /**
        * Sequence
        */
        public short seq;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;
                packet.payload.putShort(seq);
                return packet;
        }

    /**
     * Decode a mission_item_reached message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            seq = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_item_reached(){
        msgid = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_item_reached(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_ITEM_REACHED");
        //Log.d("MAVLINK_MSG_ID_MISSION_ITEM_REACHED", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_ITEM_REACHED -"+" seq:"+seq+"";
    }
}
```

```java
// MESSAGE MISSION_REQUEST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Request the information of the mission item with the sequence number seq. The response of the system to this message should be a MISSION_ITEM message. http://qgroundcontrol.org/mavlink/waypoint_protocol
*/
public class msg_mission_request extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_REQUEST = 40;
        public static final int MAVLINK_MSG_LENGTH = 4;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_REQUEST;


        /**
        * Sequence
        */
        public short seq;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_REQUEST;
                packet.payload.putShort(seq);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mission_request message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            seq = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_request(){
        msgid = MAVLINK_MSG_ID_MISSION_REQUEST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_request(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_REQUEST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_REQUEST");
        //Log.d("MAVLINK_MSG_ID_MISSION_REQUEST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_REQUEST -"+" seq:"+seq+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE MISSION_REQUEST_LIST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Request the overall list of mission items from the system/component.
*/
public class msg_mission_request_list extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_REQUEST_LIST = 43;
        public static final int MAVLINK_MSG_LENGTH = 2;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_REQUES
T_LIST;


        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_LIST;
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mission_request_list message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_request_list(){
        msgid = MAVLINK_MSG_ID_MISSION_REQUEST_LIST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_request_list(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_LIST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_REQUEST_LIST");
        //Log.d("MAVLINK_MSG_ID_MISSION_REQUEST_LIST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_REQUEST_LIST -"+" target_system:"+target_sy
stem+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE MISSION_REQUEST_PARTIAL_LIST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Request a partial list of mission items from the system/component. http://qgroundcontrol.org/mavlink/waypoint_protocol. If start and end index are the same, just send one waypoint.
*/
public class msg_mission_request_partial_list extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST = 37;
        public static final int MAVLINK_MSG_LENGTH = 6;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST;


        /**
        * Start index, 0 by default
        */
        public short start_index;
        /**
        * End index, -1 by default (-1: send list to end). Else a valid index of the list
        */
        public short end_index;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST;
                packet.payload.putShort(start_index);
                packet.payload.putShort(end_index);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mission_request_partial_list message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            start_index = payload.getShort();
            end_index = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_request_partial_list(){
        msgid = MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_request_partial_list(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_REQUEST_PARTIAL_LIST");
        //Log.d("MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST -"+" start_index:"+start_index+" end_index:"+end_index+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE MISSION_SET_CURRENT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set the mission item with sequence number seq as current item. This means that the MAV will continue to this mission item on the shortest path (not following the mission items in-between).
*/
public class msg_mission_set_current extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_SET_CURRENT = 41;
        public static final int MAVLINK_MSG_LENGTH = 4;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_SET_CURRENT;


        /**
        * Sequence
        */
        public short seq;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_SET_CURRENT;
                packet.payload.putShort(seq);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mission_set_current message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            seq = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_set_current(){
        msgid = MAVLINK_MSG_ID_MISSION_SET_CURRENT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_set_current(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_SET_CURRENT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_SET_CURRENT");
        //Log.d("MAVLINK_MSG_ID_MISSION_SET_CURRENT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_SET_CURRENT -"+" seq:"+seq+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE MISSION_WRITE_PARTIAL_LIST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* This message is sent to the MAV to write a partial list. If start index == end index, only one item will be transmitted / updated. If the start index is NOT 0 and above the current list size, this request should be REJECTED!
*/
public class msg_mission_write_partial_list extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST = 38;
        public static final int MAVLINK_MSG_LENGTH = 6;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;


        /**
        * Start index, 0 by default and smaller / equal to the largest index of the current onboard list.
        */
        public short start_index;
        /**
        * End index, equal or greater than start index.
        */
        public short end_index;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;
                packet.payload.putShort(start_index);
                packet.payload.putShort(end_index);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mission_write_partial_list message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            start_index = payload.getShort();
            end_index = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_write_partial_list(){
        msgid = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_write_partial_list(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_WRITE_PARTIAL_LIST");
        //Log.d("MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST -"+" start_index:"+start_index+" end_index:"+end_index+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE MOUNT_CONFIGURE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Message to configure a camera mount, directional antenna, etc.
*/
public class msg_mount_configure extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MOUNT_CONFIGURE = 156;
        public static final int MAVLINK_MSG_LENGTH = 6;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MOUNT_CONFIGUR
E;


        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * mount operating mode (see MAV_MOUNT_MODE enum)
        */
        public byte mount_mode;
        /**
        * (1 = yes, 0 = no)
        */
        public byte stab_roll;
        /**
        * (1 = yes, 0 = no)
        */
        public byte stab_pitch;
        /**
        * (1 = yes, 0 = no)
        */
        public byte stab_yaw;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MOUNT_CONFIGURE;
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(mount_mode);
                packet.payload.putByte(stab_roll);
                packet.payload.putByte(stab_pitch);
                packet.payload.putByte(stab_yaw);
                return packet;
        }

    /**
     * Decode a mount_configure message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            target_system = payload.getByte();
            target_component = payload.getByte();
            mount_mode = payload.getByte();
            stab_roll = payload.getByte();
            stab_pitch = payload.getByte();
            stab_yaw = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mount_configure(){
        msgid = MAVLINK_MSG_ID_MOUNT_CONFIGURE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mount_configure(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MOUNT_CONFIGURE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MOUNT_CONFIGURE");
        //Log.d("MAVLINK_MSG_ID_MOUNT_CONFIGURE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MOUNT_CONFIGURE -"+" target_system:"+target_system+
" target_component:"+target_component+" mount_mode:"+mount_mode+" stab_roll:"+stab_roll+"
 stab_pitch:"+stab_pitch+" stab_yaw:"+stab_yaw+"";
    }
}
```

```java
// MESSAGE MOUNT_CONTROL PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Message to control a camera mount, directional antenna, etc.
*/
public class msg_mount_control extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MOUNT_CONTROL = 157;
        public static final int MAVLINK_MSG_LENGTH = 15;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MOUNT_CONTROL;


        /**
        * pitch(deg*100) or lat, depending on mount mode
        */
        public int input_a;
        /**
        * roll(deg*100) or lon depending on mount mode
        */
        public int input_b;
        /**
        * yaw(deg*100) or alt (in cm) depending on mount mode
        */
        public int input_c;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * if "1" it will save current trimmed position on EEPROM (just valid for N
EUTRAL and LANDING)
        */
        public byte save_position;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MOUNT_CONTROL;
                packet.payload.putInt(input_a);
                packet.payload.putInt(input_b);
                packet.payload.putInt(input_c);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(save_position);
                return packet;
        }

    /**
     * Decode a mount_control message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            input_a = payload.getInt();
            input_b = payload.getInt();
            input_c = payload.getInt();
            target_system = payload.getByte();
            target_component = payload.getByte();
            save_position = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mount_control(){
        msgid = MAVLINK_MSG_ID_MOUNT_CONTROL;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mount_control(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MOUNT_CONTROL;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MOUNT_CONTROL");
        //Log.d("MAVLINK_MSG_ID_MOUNT_CONTROL", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MOUNT_CONTROL -"+" input_a:"+input_a+" input_b:"+in
put_b+" input_c:"+input_c+" target_system:"+target_system+" target_component:"+target_com
ponent+" save_position:"+save_position+"";
    }
}
```

```java
// MESSAGE MOUNT_STATUS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Message with some status from APM to GCS about camera or antenna mount
*/
public class msg_mount_status extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MOUNT_STATUS = 158;
        public static final int MAVLINK_MSG_LENGTH = 14;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MOUNT_STATUS;


        /**
        * pitch(deg*100) or lat, depending on mount mode
        */
        public int pointing_a;
        /**
        * roll(deg*100) or lon depending on mount mode
        */
        public int pointing_b;
        /**
        * yaw(deg*100) or alt (in cm) depending on mount mode
        */
        public int pointing_c;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MOUNT_STATUS;
                packet.payload.putInt(pointing_a);
                packet.payload.putInt(pointing_b);
                packet.payload.putInt(pointing_c);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a mount_status message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            pointing_a = payload.getInt();
            pointing_b = payload.getInt();
            pointing_c = payload.getInt();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mount_status(){
        msgid = MAVLINK_MSG_ID_MOUNT_STATUS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mount_status(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MOUNT_STATUS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MOUNT_STATUS");
        //Log.d("MAVLINK_MSG_ID_MOUNT_STATUS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MOUNT_STATUS -"+" pointing_a:"+pointing_a+" pointin
g_b:"+pointing_b+" pointing_c:"+pointing_c+" target_system:"+target_system+" target_compo
nent:"+target_component+"";
    }
}
```

```java
// MESSAGE NAMED_VALUE_FLOAT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Send a key-value pair as float. The use of this message is discouraged for normal packets, but a quite efficient way for testing new messages and getting experimental debug output.
*/
public class msg_named_value_float extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_NAMED_VALUE_FLOAT = 251;
        public static final int MAVLINK_MSG_LENGTH = 18;
        private static final long serialVersionUID = MAVLINK_MSG_ID_NAMED_VALUE_FLOAT;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * Floating point value
        */
        public float value;
        /**
        * Name of the debug variable
        */
        public byte name[] = new byte[10];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_NAMED_VALUE_FLOAT;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(value);
                 for (int i = 0; i < name.length; i++) {
                        packet.payload.putByte(name[i]);
            }
                return packet;
        }

    /**
     * Decode a named_value_float message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            value = payload.getFloat();
             for (int i = 0; i < name.length; i++) {
                        name[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_named_value_float(){
        msgid = MAVLINK_MSG_ID_NAMED_VALUE_FLOAT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_named_value_float(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_NAMED_VALUE_FLOAT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "NAMED_VALUE_FLOAT");
        //Log.d("MAVLINK_MSG_ID_NAMED_VALUE_FLOAT", toString());
    }

     /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setName(String str) {
      int len = Math.min(str.length(), 10);
      for (int i=0; i<len; i++) {
        name[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<10; i++) {                      // padding for the rest of the buffer
        name[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getName() {
                String result = "";
                for (int i = 0; i < 10; i++) {
                        if (name[i] != 0)
                                result = result + (char) name[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_NAMED_VALUE_FLOAT -"+" time_boot_ms:"+time_boot_ms+" value:"+value+" name:"+name+"";
    }
}
```

```java
// MESSAGE DEBUG PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Send a debug value. The index is used to discriminate between values. These values show up in the plot of QGroundControl as DEBUG N.
*/
public class msg_debug extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_DEBUG = 254;
        public static final int MAVLINK_MSG_LENGTH = 9;
        private static final long serialVersionUID = MAVLINK_MSG_ID_DEBUG;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * DEBUG value
        */
        public float value;
        /**
        * index of debug variable
        */
        public byte ind;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_DEBUG;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(value);
                packet.payload.putByte(ind);
                return packet;
        }

    /**
     * Decode a debug message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            value = payload.getFloat();
            ind = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_debug(){
        msgid = MAVLINK_MSG_ID_DEBUG;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_debug(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_DEBUG;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "DEBUG");
        //Log.d("MAVLINK_MSG_ID_DEBUG", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_DEBUG -"+" time_boot_ms:"+time_boot_ms+" value:"+value+" ind:"+ind+"";
    }
}
```

```java
// MESSAGE GPS_STATUS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The positioning status, as reported by GPS. This message is intended to display status information about each satellite visible to the receiver. See message GLOBAL_POSITION for the global position estimate. This message can contain information for up to 20 satellites.
*/
public class msg_gps_status extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_GPS_STATUS = 25;
        public static final int MAVLINK_MSG_LENGTH = 101;
        private static final long serialVersionUID = MAVLINK_MSG_ID_GPS_STATUS;


        /**
        * Number of satellites visible
        */
        public byte satellites_visible;
        /**
        * Global satellite ID
        */
        public byte satellite_prn[] = new byte[20];
        /**
        * 0: Satellite not used, 1: used for localization
        */
        public byte satellite_used[] = new byte[20];
        /**
        * Elevation (0: right on top of receiver, 90: on the horizon) of satellite
        */
        public byte satellite_elevation[] = new byte[20];
        /**
        * Direction of satellite, 0: 0 deg, 255: 360 deg.
        */
        public byte satellite_azimuth[] = new byte[20];
        /**
        * Signal to noise ratio of satellite
        */
        public byte satellite_snr[] = new byte[20];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_GPS_STATUS;
                packet.payload.putByte(satellites_visible);
                 for (int i = 0; i < satellite_prn.length; i++) {
                        packet.payload.putByte(satellite_prn[i]);
            }
                 for (int i = 0; i < satellite_used.length; i++) {
                        packet.payload.putByte(satellite_used[i]);
            }
                 for (int i = 0; i < satellite_elevation.length; i++) {
                        packet.payload.putByte(satellite_elevation[i]);
            }
                 for (int i = 0; i < satellite_azimuth.length; i++) {
                        packet.payload.putByte(satellite_azimuth[i]);
            }
                 for (int i = 0; i < satellite_snr.length; i++) {
                        packet.payload.putByte(satellite_snr[i]);
            }
                return packet;
        }

    /**
     * Decode a gps_status message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            satellites_visible = payload.getByte();
             for (int i = 0; i < satellite_prn.length; i++) {
                        satellite_prn[i] = payload.getByte();
                }
             for (int i = 0; i < satellite_used.length; i++) {
                        satellite_used[i] = payload.getByte();
                }
             for (int i = 0; i < satellite_elevation.length; i++) {
                        satellite_elevation[i] = payload.getByte();
                }
             for (int i = 0; i < satellite_azimuth.length; i++) {
                        satellite_azimuth[i] = payload.getByte();
                }
             for (int i = 0; i < satellite_snr.length; i++) {
                        satellite_snr[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_gps_status(){
        msgid = MAVLINK_MSG_ID_GPS_STATUS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_gps_status(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_GPS_STATUS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "GPS_STATUS");
        //Log.d("MAVLINK_MSG_ID_GPS_STATUS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_GPS_STATUS -"+" satellites_visible:"+satellites_visible+" satellite_prn:"+satellite_prn+" satellite_used:"+satellite_used+" satellite_elevation:"+satellite_elevation+" satellite_azimuth:"+satellite_azimuth+" satellite_snr:"+satellite_snr+"";
    }
}
```

```java
// MESSAGE MISSION_ACK PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Ack message during MISSION handling. The type field states if this message is a positive ack (type=0) or if an error happened (type=non-zero).
*/
public class msg_mission_ack extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_MISSION_ACK = 47;
        public static final int MAVLINK_MSG_LENGTH = 3;
        private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_ACK;


        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * See MAV_MISSION_RESULT enum
        */
        public byte type;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_MISSION_ACK;
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(type);
                return packet;
        }

    /**
     * Decode a mission_ack message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            target_system = payload.getByte();
            target_component = payload.getByte();
            type = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_mission_ack(){
        msgid = MAVLINK_MSG_ID_MISSION_ACK;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_mission_ack(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_MISSION_ACK;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "MISSION_ACK");
        //Log.d("MAVLINK_MSG_ID_MISSION_ACK", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_MISSION_ACK -"+" target_system:"+target_system+" target_component:"+target_component+" type:"+type+"";
    }
}
```

```java
// MESSAGE NAMED_VALUE_INT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Send a key-value pair as integer. The use of this message is discouraged for normal packets, but a quite efficient way for testing new messages and getting experimental debug output.
*/
public class msg_named_value_int extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_NAMED_VALUE_INT = 252;
        public static final int MAVLINK_MSG_LENGTH = 18;
        private static final long serialVersionUID = MAVLINK_MSG_ID_NAMED_VALUE_INT;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * Signed integer value
        */
        public int value;
        /**
        * Name of the debug variable
        */
        public byte name[] = new byte[10];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_NAMED_VALUE_INT;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putInt(value);
                 for (int i = 0; i < name.length; i++) {
                        packet.payload.putByte(name[i]);
            }
                return packet;
        }

    /**
     * Decode a named_value_int message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            value = payload.getInt();
             for (int i = 0; i < name.length; i++) {
                        name[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_named_value_int(){
        msgid = MAVLINK_MSG_ID_NAMED_VALUE_INT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_named_value_int(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_NAMED_VALUE_INT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "NAMED_VALUE_INT");
        //Log.d("MAVLINK_MSG_ID_NAMED_VALUE_INT", toString());
    }

     /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setName(String str) {
      int len = Math.min(str.length(), 10);
      for (int i=0; i<len; i++) {
        name[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<10; i++) {                      // padding for the rest of the buffer
        name[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getName() {
                String result = "";
                for (int i = 0; i < 10; i++) {
                        if (name[i] != 0)
                                result = result + (char) name[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_NAMED_VALUE_INT -"+" time_boot_ms:"+time_boot_ms+" value:"+value+" name:"+name+"";
    }
}
```

```java
// MESSAGE ROLL_PITCH_YAW_RATES_THRUST_SETPOINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Setpoint in roll, pitch, yaw rates and thrust currently active on the system.
*/
public class msg_roll_pitch_yaw_rates_thrust_setpoint extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT = 80;
        public static final int MAVLINK_MSG_LENGTH = 20;
        private static final long serialVersionUID = MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT;


        /**
        * Timestamp in milliseconds since system boot
        */
        public int time_boot_ms;
        /**
        * Desired roll rate in radians per second
        */
        public float roll_rate;
        /**
        * Desired pitch rate in radians per second
        */
        public float pitch_rate;
        /**
        * Desired yaw rate in radians per second
        */
        public float yaw_rate;
        /**
        * Collective thrust, normalized to 0 .. 1
        */
        public float thrust;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(roll_rate);
                packet.payload.putFloat(pitch_rate);
                packet.payload.putFloat(yaw_rate);
                packet.payload.putFloat(thrust);
                return packet;
        }

    /**
     * Decode a roll_pitch_yaw_rates_thrust_setpoint message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            roll_rate = payload.getFloat();
            pitch_rate = payload.getFloat();
            yaw_rate = payload.getFloat();
            thrust = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_roll_pitch_yaw_rates_thrust_setpoint(){
        msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_roll_pitch_yaw_rates_thrust_setpoint(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "ROLL_PITCH_YAW_RATES_THRUST_SETPOINT");
        //Log.d("MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT -"+" time_boot_ms:"+time_boot_ms+" roll_rate:"+roll_rate+" pitch_rate:"+pitch_rate+" yaw_rate:"+yaw_rate+" thrust:"+thrust+"";
    }
}
```

```java
// MESSAGE SET_MODE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set the system mode, as defined by enum MAV_MODE. There is no target component id as the mode is by definition for the overall aircraft, not only for one component.
*/
public class msg_set_mode extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_MODE = 11;
        public static final int MAVLINK_MSG_LENGTH = 6;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_MODE;


        /**
        * The new autopilot-specific mode. This field can be ignored by an autopilot.
        */
        public int custom_mode;
        /**
        * The system setting the mode
        */
        public byte target_system;
        /**
        * The new base mode
        */
        public byte base_mode;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_MODE;
                packet.payload.putInt(custom_mode);
                packet.payload.putByte(target_system);
                packet.payload.putByte(base_mode);
                return packet;
        }

    /**
     * Decode a set_mode message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            custom_mode = payload.getInt();
            target_system = payload.getByte();
            base_mode = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_mode(){
        msgid = MAVLINK_MSG_ID_SET_MODE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_mode(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_MODE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_MODE");
        //Log.d("MAVLINK_MSG_ID_SET_MODE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_MODE -"+" custom_mode:"+custom_mode+" target_system:"+target_system+" base_mode:"+base_mode+"";
    }
}
```

```java
// MESSAGE NAV_CONTROLLER_OUTPUT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Outputs of the APM navigation controller. The primary use of this message is to check the response and signs of the controller before actual flight and to assist with tuning controller parameters.
*/
public class msg_nav_controller_output extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT = 62;
        public static final int MAVLINK_MSG_LENGTH = 26;
        private static final long serialVersionUID = MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT;


        /**
        * Current desired roll in degrees
        */
        public float nav_roll;
        /**
        * Current desired pitch in degrees
        */
        public float nav_pitch;
        /**
        * Current altitude error in meters
        */
        public float alt_error;
        /**
        * Current airspeed error in meters/second
        */
        public float aspd_error;
        /**
        * Current crosstrack error on x-y plane in meters
        */
        public float xtrack_error;
        /**
        * Current desired heading in degrees
        */
        public short nav_bearing;
        /**
        * Bearing to current MISSION/target in degrees
        */
        public short target_bearing;
        /**
        * Distance to active MISSION in meters
        */
        public short wp_dist;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT;
                packet.payload.putFloat(nav_roll);
                packet.payload.putFloat(nav_pitch);
                packet.payload.putFloat(alt_error);
                packet.payload.putFloat(aspd_error);
                packet.payload.putFloat(xtrack_error);
                packet.payload.putShort(nav_bearing);
                packet.payload.putShort(target_bearing);
                packet.payload.putShort(wp_dist);
                return packet;
        }

    /**
     * Decode a nav_controller_output message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            nav_roll = payload.getFloat();
            nav_pitch = payload.getFloat();
            alt_error = payload.getFloat();
            aspd_error = payload.getFloat();
            xtrack_error = payload.getFloat();
            nav_bearing = payload.getShort();
            target_bearing = payload.getShort();
            wp_dist = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_nav_controller_output(){
        msgid = MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_nav_controller_output(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "NAV_CONTROLLER_OUTPUT");
        //Log.d("MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT -"+" nav_roll:"+nav_roll+" nav_pitch:"+nav_pitch+" alt_error:"+alt_error+" aspd_error:"+aspd_error+" xtrack_error:"+xtrack_error+" nav_bearing:"+nav_bearing+" target_bearing:"+target_bearing+" wp_dist:"+wp_dist+"";
    }
}
```

```java
// MESSAGE OPTICAL_FLOW PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Optical flow from a flow sensor (e.g. optical mouse sensor)
*/
public class msg_optical_flow extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_OPTICAL_FLOW = 100;
        public static final int MAVLINK_MSG_LENGTH = 26;
        private static final long serialVersionUID = MAVLINK_MSG_ID_OPTICAL_FLOW;


        /**
        * Timestamp (UNIX)
        */
        public long time_usec;
        /**
        * Flow in meters in x-sensor direction, angular-speed compensated
        */
        public float flow_comp_m_x;
        /**
        * Flow in meters in y-sensor direction, angular-speed compensated
        */
        public float flow_comp_m_y;
        /**
        * Ground distance in meters. Positive value: distance known. Negative value: Unknown distance
        */
        public float ground_distance;
        /**
        * Flow in pixels in x-sensor direction
        */
        public short flow_x;
        /**
        * Flow in pixels in y-sensor direction
        */
        public short flow_y;
        /**
        * Sensor ID
        */
        public byte sensor_id;
        /**
        * Optical flow quality / confidence. 0: bad, 255: maximum quality
        */
        public byte quality;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_OPTICAL_FLOW;
                packet.payload.putLong(time_usec);
                packet.payload.putFloat(flow_comp_m_x);
                packet.payload.putFloat(flow_comp_m_y);
                packet.payload.putFloat(ground_distance);
                packet.payload.putShort(flow_x);
                packet.payload.putShort(flow_y);
                packet.payload.putByte(sensor_id);
                packet.payload.putByte(quality);
                return packet;
        }

    /**
     * Decode a optical_flow message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            flow_comp_m_x = payload.getFloat();
            flow_comp_m_y = payload.getFloat();
            ground_distance = payload.getFloat();
            flow_x = payload.getShort();
            flow_y = payload.getShort();
            sensor_id = payload.getByte();
            quality = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_optical_flow(){
        msgid = MAVLINK_MSG_ID_OPTICAL_FLOW;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_optical_flow(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_OPTICAL_FLOW;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "OPTICAL_FLOW");
        //Log.d("MAVLINK_MSG_ID_OPTICAL_FLOW", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_OPTICAL_FLOW -"+" time_usec:"+time_usec+" flow_comp_m_x:"+flow_comp_m_x+" flow_comp_m_y:"+flow_comp_m_y+" ground_distance:"+ground_distance+" flow_x:"+flow_x+" flow_y:"+flow_y+" sensor_id:"+sensor_id+" quality:"+quality+"";
    }
}
```

```java
// MESSAGE PARAM_REQUEST_LIST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Request all parameters of this component. After his request, all parameters are emitted.
*/
public class msg_param_request_list extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_PARAM_REQUEST_LIST = 21;
        public static final int MAVLINK_MSG_LENGTH = 2;
        private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_REQUEST_LIST;


        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_LIST;
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a param_request_list message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_param_request_list(){
        msgid = MAVLINK_MSG_ID_PARAM_REQUEST_LIST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_param_request_list(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_LIST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "PARAM_REQUEST_LIST");
        //Log.d("MAVLINK_MSG_ID_PARAM_REQUEST_LIST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_PARAM_REQUEST_LIST -"+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE PARAM_REQUEST_READ PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Request to read the onboard parameter with the param_id string id. Onboard parameters are stored as key[const char*] -> value[float]. This allows to send a parameter to any other component (such as the GCS) without the need of previous knowledge of possible parameter names. Thus the same GCS can store different parameters for different autopilots. See also http://qgroundcontrol.org/parameter_interface for a full documentation of QGroundControl and IMU code.
*/
public class msg_param_request_read extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_PARAM_REQUEST_READ = 20;
        public static final int MAVLINK_MSG_LENGTH = 20;
        private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_REQUEST_READ;


        /**
        * Parameter index. Send -1 to use the param ID field as identifier (else the param id will be ignored)
        */
        public short param_index;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * Onboard parameter id, terminated by NULL if the length is less than 16 human-readable chars and WITHOUT null termination (NULL) byte if the length is exactly 16 chars - applications have to provide 16+1 bytes storage if the ID is stored as string
        */
        public byte param_id[] = new byte[16];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_READ;
                packet.payload.putShort(param_index);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                 for (int i = 0; i < param_id.length; i++) {
                        packet.payload.putByte(param_id[i]);
            }
                return packet;
        }

    /**
     * Decode a param_request_read message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            param_index = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
             for (int i = 0; i < param_id.length; i++) {
                        param_id[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_param_request_read(){
        msgid = MAVLINK_MSG_ID_PARAM_REQUEST_READ;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_param_request_read(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_READ;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "PARAM_REQUEST_READ");
        //Log.d("MAVLINK_MSG_ID_PARAM_REQUEST_READ", toString());
    }

      /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setParam_Id(String str) {
      int len = Math.min(str.length(), 16);
      for (int i=0; i<len; i++) {
        param_id[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<16; i++) {                    // padding for the rest of the buffer
        param_id[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getParam_Id() {
                String result = "";
                for (int i = 0; i < 16; i++) {
                        if (param_id[i] != 0)
                                result = result + (char) param_id[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_PARAM_REQUEST_READ -"+" param_index:"+param_index+" target_system:"+target_system+" target_component:"+target_component+" param_id:"+param_id+"";
    }
}
```

```java
// MESSAGE PARAM_SET PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set a parameter value TEMPORARILY to RAM. It will be reset to default on system reboot. Send the ACTION MAV_ACTION_STORAGE_WRITE to PERMANENTLY write the RAM contents to EEPROM. IMPORTANT: The receiving component should acknowledge the new parameter value by sending a param_value message to all communication partners. This will also ensure that multiple GCS all have an up-to-date list of all parameters. If the sending GCS did not receive a PARAM_VALUE message within its timeout time, it should re-send the PARAM_SET message.
*/
public class msg_param_set extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_PARAM_SET = 23;
        public static final int MAVLINK_MSG_LENGTH = 23;
        private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_SET;


        /**
        * Onboard parameter value
        */
        public float param_value;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * Onboard parameter id, terminated by NULL if the length is less than 16 human-readable chars and WITHOUT null termination (NULL) byte if the length is exactly 16 chars - applications have to provide 16+1 bytes storage if the ID is stored as string
        */
        public byte param_id[] = new byte[16];
        /**
        * Onboard parameter type: see the MAV_PARAM_TYPE enum for supported data types.
        */
        public byte param_type;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_PARAM_SET;
                packet.payload.putFloat(param_value);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                 for (int i = 0; i < param_id.length; i++) {
                        packet.payload.putByte(param_id[i]);
            }
                packet.payload.putByte(param_type);
                return packet;
        }

    /**
     * Decode a param_set message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            param_value = payload.getFloat();
            target_system = payload.getByte();
            target_component = payload.getByte();
             for (int i = 0; i < param_id.length; i++) {
                        param_id[i] = payload.getByte();
                }
            param_type = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_param_set(){
        msgid = MAVLINK_MSG_ID_PARAM_SET;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_param_set(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_PARAM_SET;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "PARAM_SET");
        //Log.d("MAVLINK_MSG_ID_PARAM_SET", toString());
    }

       /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setParam_Id(String str) {
      int len = Math.min(str.length(), 16);
      for (int i=0; i<len; i++) {
        param_id[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<16; i++) {                      // padding for the rest of the buffer
        param_id[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getParam_Id() {
                String result = "";
                for (int i = 0; i < 16; i++) {
                        if (param_id[i] != 0)
                                result = result + (char) param_id[i];
                        else
                                break;
                }
                return result;

        }
    /**
```

```java
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_PARAM_SET -"+" param_value:"+param_value+" target_system:"+target_system+" target_component:"+target_component+" param_id:"+param_id+" param_type:"+param_type+"";
    }
}
```

```java
// MESSAGE PARAM_VALUE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Emit the value of a onboard parameter. The inclusion of param_count and param_index in the message allows the recipient to keep track of received parameters and allows him to re-request missing parameters after a loss or timeout.
*/
public class msg_param_value extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_PARAM_VALUE = 22;
        public static final int MAVLINK_MSG_LENGTH = 25;
        private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_VALUE;


        /**
        * Onboard parameter value
        */
        public float param_value;
        /**
        * Total number of onboard parameters
        */
        public short param_count;
        /**
        * Index of this onboard parameter
        */
        public short param_index;
        /**
        * Onboard parameter id, terminated by NULL if the length is less than 16 human-readable chars and WITHOUT null termination (NULL) byte if the length is exactly 16 chars - applications have to provide 16+1 bytes storage if the ID is stored as string
        */
        public byte param_id[] = new byte[16];
        /**
        * Onboard parameter type: see the MAV_PARAM_TYPE enum for supported data types.
        */
        public byte param_type;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_PARAM_VALUE;
                packet.payload.putFloat(param_value);
                packet.payload.putShort(param_count);
                packet.payload.putShort(param_index);
                 for (int i = 0; i < param_id.length; i++) {
                        packet.payload.putByte(param_id[i]);
            }
                packet.payload.putByte(param_type);
                return packet;
        }

    /**
     * Decode a param_value message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            param_value = payload.getFloat();
            param_count = payload.getShort();
            param_index = payload.getShort();
             for (int i = 0; i < param_id.length; i++) {
                        param_id[i] = payload.getByte();
                }
            param_type = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_param_value(){
        msgid = MAVLINK_MSG_ID_PARAM_VALUE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_param_value(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_PARAM_VALUE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "PARAM_VALUE");
        //Log.d("MAVLINK_MSG_ID_PARAM_VALUE", toString());
    }

       /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setParam_Id(String str) {
      int len = Math.min(str.length(), 16);
      for (int i=0; i<len; i++) {
        param_id[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<16; i++) {                      // padding for the rest of the buffer
        param_id[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getParam_Id() {
                String result = "";
                for (int i = 0; i < 16; i++) {
                        if (param_id[i] != 0)
                                result = result + (char) param_id[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_PARAM_VALUE -"+" param_value:"+param_value+" param_
count:"+param_count+" param_index:"+param_index+" param_id:"+param_id+" param_type:"+para
m_type+"";
    }
}
```

```java
// MESSAGE PING PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* A ping message either requesting or responding to a ping. This allows to measure the system latencies, including serial port, radio modem and UDP connections.
*/
public class msg_ping extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_PING = 4;
        public static final int MAVLINK_MSG_LENGTH = 14;
        private static final long serialVersionUID = MAVLINK_MSG_ID_PING;


        /**
        * Unix timestamp in microseconds
        */
        public long time_usec;
        /**
        * PING sequence
        */
        public int seq;
        /**
        * 0: request ping from all receiving systems, if greater than 0: message is a ping response and number is the system id of the requesting system
        */
        public byte target_system;
        /**
        * 0: request ping from all receiving components, if greater than 0: message is a ping response and number is the system id of the requesting system
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_PING;
                packet.payload.putLong(time_usec);
                packet.payload.putInt(seq);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a ping message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            seq = payload.getInt();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_ping(){
        msgid = MAVLINK_MSG_ID_PING;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_ping(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_PING;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "PING");
        //Log.d("MAVLINK_MSG_ID_PING", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_PING -"+" time_usec:"+time_usec+" seq:"+seq+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE RADIO PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Status generated by radio
*/
public class msg_radio extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_RADIO = 166;
        public static final int MAVLINK_MSG_LENGTH = 9;
        private static final long serialVersionUID = MAVLINK_MSG_ID_RADIO;


        /**
        * receive errors
        */
        public short rxerrors;
        /**
        * count of error corrected packets
        */
        public short fixed;
        /**
        * local signal strength
        */
        public byte rssi;
        /**
        * remote signal strength
        */
        public byte remrssi;
        /**
        * how full the tx buffer is as a percentage
        */
        public byte txbuf;
        /**
        * background noise level
        */
        public byte noise;
        /**
        * remote background noise level
        */
        public byte remnoise;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_RADIO;
                packet.payload.putShort(rxerrors);
                packet.payload.putShort(fixed);
                packet.payload.putByte(rssi);
                packet.payload.putByte(remrssi);
                packet.payload.putByte(txbuf);
                packet.payload.putByte(noise);
                packet.payload.putByte(remnoise);
                return packet;
        }

    /**
     * Decode a radio message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            rxerrors = payload.getShort();
            fixed = payload.getShort();
            rssi = payload.getByte();
            remrssi = payload.getByte();
            txbuf = payload.getByte();
            noise = payload.getByte();
            remnoise = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_radio(){
        msgid = MAVLINK_MSG_ID_RADIO;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_radio(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_RADIO;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "RADIO");
        //Log.d("MAVLINK_MSG_ID_RADIO", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_RADIO -"+" rxerrors:"+rxerrors+" fixed:"+fixed+" rs
si:"+rssi+" remrssi:"+remrssi+" txbuf:"+txbuf+" noise:"+noise+" remnoise:"+remnoise+"";
    }
}
```

```java
// MESSAGE RAW_IMU PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The RAW IMU readings for the usual 9DOF sensor setup. This message should always contain the true raw values without any scaling to allow data capture and system debugging.
*/
public class msg_raw_imu extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_RAW_IMU = 27;
        public static final int MAVLINK_MSG_LENGTH = 26;
        private static final long serialVersionUID = MAVLINK_MSG_ID_RAW_IMU;


        /**
        * Timestamp (microseconds since UNIX epoch or microseconds since system boot)
        */
        public long time_usec;
        /**
        * X acceleration (raw)
        */
        public short xacc;
        /**
        * Y acceleration (raw)
        */
        public short yacc;
        /**
        * Z acceleration (raw)
        */
        public short zacc;
        /**
        * Angular speed around X axis (raw)
        */
        public short xgyro;
        /**
        * Angular speed around Y axis (raw)
        */
        public short ygyro;
        /**
        * Angular speed around Z axis (raw)
        */
        public short zgyro;
        /**
        * X Magnetic field (raw)
        */
        public short xmag;
        /**
        * Y Magnetic field (raw)
        */
        public short ymag;
        /**
        * Z Magnetic field (raw)
        */
        public short zmag;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_RAW_IMU;
                packet.payload.putLong(time_usec);
                packet.payload.putShort(xacc);
                packet.payload.putShort(yacc);
                packet.payload.putShort(zacc);
                packet.payload.putShort(xgyro);
                packet.payload.putShort(ygyro);
                packet.payload.putShort(zgyro);
                packet.payload.putShort(xmag);
                packet.payload.putShort(ymag);
                packet.payload.putShort(zmag);
                return packet;
        }

    /**
     * Decode a raw_imu message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            xacc = payload.getShort();
            yacc = payload.getShort();
            zacc = payload.getShort();
            xgyro = payload.getShort();
            ygyro = payload.getShort();
            zgyro = payload.getShort();
            xmag = payload.getShort();
            ymag = payload.getShort();
            zmag = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_raw_imu(){
        msgid = MAVLINK_MSG_ID_RAW_IMU;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_raw_imu(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_RAW_IMU;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "RAW_IMU");
        //Log.d("MAVLINK_MSG_ID_RAW_IMU", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_RAW_IMU -"+" time_usec:"+time_usec+" xacc:"+xacc+" yacc:"+yacc+" zacc:"+zacc+" xgyro:"+xgyro+" ygyro:"+ygyro+" zgyro:"+zgyro+" xmag:"+xmag+" ymag:"+ymag+" zmag:"+zmag+"";
    }
}
```

```java
// MESSAGE RAW_PRESSURE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The RAW pressure readings for the typical setup of one absolute pressure and one differential pressure sensor. The sensor values should be the raw, UNSCALED ADC values.
*/
public class msg_raw_pressure extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_RAW_PRESSURE = 28;
        public static final int MAVLINK_MSG_LENGTH = 16;
        private static final long serialVersionUID = MAVLINK_MSG_ID_RAW_PRESSURE;


        /**
        * Timestamp (microseconds since UNIX epoch or microseconds since system boot)
        */
        public long time_usec;
        /**
        * Absolute pressure (raw)
        */
        public short press_abs;
        /**
        * Differential pressure 1 (raw)
        */
        public short press_diff1;
        /**
        * Differential pressure 2 (raw)
        */
        public short press_diff2;
        /**
        * Raw Temperature measurement (raw)
        */
        public short temperature;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_RAW_PRESSURE;
                packet.payload.putLong(time_usec);
                packet.payload.putShort(press_abs);
                packet.payload.putShort(press_diff1);
                packet.payload.putShort(press_diff2);
                packet.payload.putShort(temperature);
                return packet;
        }

    /**
     * Decode a raw_pressure message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getLong();
            press_abs = payload.getShort();
            press_diff1 = payload.getShort();
            press_diff2 = payload.getShort();
            temperature = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_raw_pressure(){
        msgid = MAVLINK_MSG_ID_RAW_PRESSURE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_raw_pressure(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_RAW_PRESSURE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "RAW_PRESSURE");
        //Log.d("MAVLINK_MSG_ID_RAW_PRESSURE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_RAW_PRESSURE -"+" time_usec:"+time_usec+" press_abs:"+press_abs+" press_diff1:"+press_diff1+" press_diff2:"+press_diff2+" temperature:"+temperature+"";
    }
}
```

```java
// MESSAGE RC_CHANNELS_OVERRIDE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The RAW values of the RC channels sent to the MAV to override info received from the RC radio. A value of -1 means no change to that channel. A value of 0 means control of that channel should be released back to the RC radio. The standard PPM modulation is as follows: 1000 microseconds: 0%, 2000 microseconds: 100%. Individual receivers/transmitters might violate this specification.
*/
public class msg_rc_channels_override extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE = 70;
        public static final int MAVLINK_MSG_LENGTH = 18;
        private static final long serialVersionUID = MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE;


        /**
        * RC channel 1 value, in microseconds
        */
        public short chan1_raw;
        /**
        * RC channel 2 value, in microseconds
        */
        public short chan2_raw;
        /**
        * RC channel 3 value, in microseconds
        */
        public short chan3_raw;
        /**
        * RC channel 4 value, in microseconds
        */
        public short chan4_raw;
        /**
        * RC channel 5 value, in microseconds
        */
        public short chan5_raw;
        /**
        * RC channel 6 value, in microseconds
        */
        public short chan6_raw;
        /**
        * RC channel 7 value, in microseconds
        */
        public short chan7_raw;
        /**
        * RC channel 8 value, in microseconds
        */
        public short chan8_raw;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE;
                packet.payload.putShort(chan1_raw);
                packet.payload.putShort(chan2_raw);
                packet.payload.putShort(chan3_raw);
                packet.payload.putShort(chan4_raw);
                packet.payload.putShort(chan5_raw);
                packet.payload.putShort(chan6_raw);
                packet.payload.putShort(chan7_raw);
                packet.payload.putShort(chan8_raw);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a rc_channels_override message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            chan1_raw = payload.getShort();
            chan2_raw = payload.getShort();
            chan3_raw = payload.getShort();
            chan4_raw = payload.getShort();
            chan5_raw = payload.getShort();
            chan6_raw = payload.getShort();
            chan7_raw = payload.getShort();
            chan8_raw = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_rc_channels_override(){
        msgid = MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_rc_channels_override(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "RC_CHANNELS_OVERRIDE");
        //Log.d("MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE -"+" chan1_raw:"+chan1_raw+" chan2_raw:"+chan2_raw+" chan3_raw:"+chan3_raw+" chan4_raw:"+chan4_raw+" chan5_raw:"+chan5_
raw+" chan6_raw:"+chan6_raw+" chan7_raw:"+chan7_raw+" chan8_raw:"+chan8_raw+" target_syst
em:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE RC_CHANNELS_RAW PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The RAW values of the RC channels received. The standard PPM modulation is as follows: 1000 microseconds: 0%, 2000 microseconds: 100%. Individual receivers/transmitters might violate this specification.
*/
public class msg_rc_channels_raw extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_RC_CHANNELS_RAW = 35;
        public static final int MAVLINK_MSG_LENGTH = 22;
        private static final long serialVersionUID = MAVLINK_MSG_ID_RC_CHANNELS_RAW;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * RC channel 1 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan1_raw;
        /**
        * RC channel 2 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan2_raw;
        /**
        * RC channel 3 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan3_raw;
        /**
        * RC channel 4 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan4_raw;
        /**
        * RC channel 5 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan5_raw;
        /**
        * RC channel 6 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan6_raw;
        /**
        * RC channel 7 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan7_raw;
        /**
        * RC channel 8 value, in microseconds. A value of 65535 implies the channel is unused.
        */
        public short chan8_raw;
        /**
        * Servo output port (set of 8 outputs = 1 port). Most MAVs will just use one, but this allows for more than 8 servos.
        */
        public byte port;
        /**
        * Receive signal strength indicator, 0: 0%, 100: 100%, 255: invalid/unknown.
        */
        public byte rssi;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_RC_CHANNELS_RAW;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putShort(chan1_raw);
                packet.payload.putShort(chan2_raw);
                packet.payload.putShort(chan3_raw);
                packet.payload.putShort(chan4_raw);
                packet.payload.putShort(chan5_raw);
                packet.payload.putShort(chan6_raw);
                packet.payload.putShort(chan7_raw);
                packet.payload.putShort(chan8_raw);
                packet.payload.putByte(port);
                packet.payload.putByte(rssi);
                return packet;
        }

    /**
     * Decode a rc_channels_raw message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            chan1_raw = payload.getShort();
            chan2_raw = payload.getShort();
            chan3_raw = payload.getShort();
            chan4_raw = payload.getShort();
            chan5_raw = payload.getShort();
            chan6_raw = payload.getShort();
            chan7_raw = payload.getShort();
            chan8_raw = payload.getShort();
            port = payload.getByte();
            rssi = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_rc_channels_raw(){
        msgid = MAVLINK_MSG_ID_RC_CHANNELS_RAW;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_rc_channels_raw(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_RC_CHANNELS_RAW;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "RC_CHANNELS_RAW");
        //Log.d("MAVLINK_MSG_ID_RC_CHANNELS_RAW", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_RC_CHANNELS_RAW -"+" time_boot_ms:"+time_boot_ms+" chan1_raw:"+chan1_raw+" chan2_raw:"+chan2_raw+" chan3_raw:"+chan3_raw+" chan4_raw:"+chan4_raw+" chan5_raw:"+chan5_raw+" chan6_raw:"+chan6_raw+" chan7_raw:"+chan7_raw+" chan8_raw:"+chan8_raw+" port:"+port+" rssi:"+rssi+"";
    }
}
```

```java
// MESSAGE RC_CHANNELS_SCALED PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The scaled values of the RC channels received. (-100%) -10000, (0%) 0, (100%) 10000. Channels that are inactive should be set to 65535.
*/
public class msg_rc_channels_scaled extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_RC_CHANNELS_SCALED = 34;
        public static final int MAVLINK_MSG_LENGTH = 22;
        private static final long serialVersionUID = MAVLINK_MSG_ID_RC_CHANNELS_SCALED;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * RC channel 1 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan1_scaled;
        /**
        * RC channel 2 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan2_scaled;
        /**
        * RC channel 3 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan3_scaled;
        /**
        * RC channel 4 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan4_scaled;
        /**
        * RC channel 5 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan5_scaled;
        /**
        * RC channel 6 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan6_scaled;
        /**
        * RC channel 7 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan7_scaled;
        /**
        * RC channel 8 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invalid) 32767.
        */
        public short chan8_scaled;
        /**
        * Servo output port (set of 8 outputs = 1 port). Most MAVs will just use one, but this allows for more than 8 servos.
    */
    public byte port;
    /**
    * Receive signal strength indicator, 0: 0%, 100: 100%, 255: invalid/unknown.
    */
    public byte rssi;

    /**
     * Generates the payload for a mavlink message for a message of this type
     * @return
     */
    public MAVLinkPacket pack(){
            MAVLinkPacket packet = new MAVLinkPacket();
            packet.len = MAVLINK_MSG_LENGTH;
            packet.sysid = 255;
            packet.compid = 190;
            packet.msgid = MAVLINK_MSG_ID_RC_CHANNELS_SCALED;
            packet.payload.putInt(time_boot_ms);
            packet.payload.putShort(chan1_scaled);
            packet.payload.putShort(chan2_scaled);
            packet.payload.putShort(chan3_scaled);
            packet.payload.putShort(chan4_scaled);
            packet.payload.putShort(chan5_scaled);
            packet.payload.putShort(chan6_scaled);
            packet.payload.putShort(chan7_scaled);
            packet.payload.putShort(chan8_scaled);
            packet.payload.putByte(port);
            packet.payload.putByte(rssi);
            return packet;
    }

/**
 * Decode a rc_channels_scaled message into this class fields
 *
 * @param payload The message to decode
 */
public void unpack(MAVLinkPayload payload) {
    payload.resetIndex();
        time_boot_ms = payload.getInt();
        chan1_scaled = payload.getShort();
        chan2_scaled = payload.getShort();
        chan3_scaled = payload.getShort();
        chan4_scaled = payload.getShort();
        chan5_scaled = payload.getShort();
        chan6_scaled = payload.getShort();
        chan7_scaled = payload.getShort();
        chan8_scaled = payload.getShort();
        port = payload.getByte();
        rssi = payload.getByte();
}

 /**
 * Constructor for a new message, just initializes the msgid
 */
public msg_rc_channels_scaled(){
    msgid = MAVLINK_MSG_ID_RC_CHANNELS_SCALED;
}

/**
 * Constructor for a new message, initializes the message with the payload
 * from a mavlink packet
 *
 */
public msg_rc_channels_scaled(MAVLinkPacket mavLinkPacket){
    this.sysid = mavLinkPacket.sysid;
    this.compid = mavLinkPacket.compid;
```

```java
        this.msgid = MAVLINK_MSG_ID_RC_CHANNELS_SCALED;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "RC_CHANNELS_SCALED");
        //Log.d("MAVLINK_MSG_ID_RC_CHANNELS_SCALED", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_RC_CHANNELS_SCALED -"+" time_boot_ms:"+time_boot_ms+" chan1_scaled:"+chan1_scaled+" chan2_scaled:"+chan2_scaled+" chan3_scaled:"+chan3_scaled+" chan4_scaled:"+chan4_scaled+" chan5_scaled:"+chan5_scaled+" chan6_scaled:"+chan6_scaled+" chan7_scaled:"+chan7_scaled+" chan8_scaled:"+chan8_scaled+" port:"+port+" rssi:"+rssi+"";
    }
}
```

```java
// MESSAGE REQUEST_DATA_STREAM PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
*
*/
public class msg_request_data_stream extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_REQUEST_DATA_STREAM = 66;
        public static final int MAVLINK_MSG_LENGTH = 6;
        private static final long serialVersionUID = MAVLINK_MSG_ID_REQUEST_DATA_STREAM;


        /**
        * The requested interval between two messages of this type
        */
        public short req_message_rate;
        /**
        * The target requested to send the message stream.
        */
        public byte target_system;
        /**
        * The target requested to send the message stream.
        */
        public byte target_component;
        /**
        * The ID of the requested data stream
        */
        public byte req_stream_id;
        /**
        * 1 to start sending, 0 to stop sending.
        */
        public byte start_stop;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_REQUEST_DATA_STREAM;
                packet.payload.putShort(req_message_rate);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(req_stream_id);
                packet.payload.putByte(start_stop);
                return packet;
        }

    /**
     * Decode a request_data_stream message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            req_message_rate = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
            req_stream_id = payload.getByte();
            start_stop = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_request_data_stream(){
        msgid = MAVLINK_MSG_ID_REQUEST_DATA_STREAM;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_request_data_stream(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_REQUEST_DATA_STREAM;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "REQUEST_DATA_STREAM");
        //Log.d("MAVLINK_MSG_ID_REQUEST_DATA_STREAM", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_REQUEST_DATA_STREAM -"+" req_message_rate:"+req_message_rate+" target_system:"+target_system+" target_component:"+target_component+" req_stream_id:"+req_stream_id+" start_stop:"+start_stop+"";
    }
}
```

```java
// MESSAGE ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Setpoint in rollspeed, pitchspeed, yawspeed currently active on the system.
*/
public class msg_roll_pitch_yaw_speed_thrust_setpoint extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT = 59;
        public static final int MAVLINK_MSG_LENGTH = 20;
        private static final long serialVersionUID = MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT;


        /**
        * Timestamp in milliseconds since system boot
        */
        public int time_boot_ms;
        /**
        * Desired roll angular speed in rad/s
        */
        public float roll_speed;
        /**
        * Desired pitch angular speed in rad/s
        */
        public float pitch_speed;
        /**
        * Desired yaw angular speed in rad/s
        */
        public float yaw_speed;
        /**
        * Collective thrust, normalized to 0 .. 1
        */
        public float thrust;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(roll_speed);
                packet.payload.putFloat(pitch_speed);
                packet.payload.putFloat(yaw_speed);
                packet.payload.putFloat(thrust);
                return packet;
        }

    /**
     * Decode a roll_pitch_yaw_speed_thrust_setpoint message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            roll_speed = payload.getFloat();
            pitch_speed = payload.getFloat();
            yaw_speed = payload.getFloat();
            thrust = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_roll_pitch_yaw_speed_thrust_setpoint(){
        msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_roll_pitch_yaw_speed_thrust_setpoint(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT");
        //Log.d("MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT -"+" time_boot_ms:"+time_boot_ms+" roll_speed:"+roll_speed+" pitch_speed:"+pitch_speed+" yaw_speed:"+yaw_speed+" thrust:"+thrust+"";
    }
}
```

```java
// MESSAGE ROLL_PITCH_YAW_THRUST_SETPOINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Setpoint in roll, pitch, yaw currently active on the system.
*/
public class msg_roll_pitch_yaw_thrust_setpoint extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT = 58;
        public static final int MAVLINK_MSG_LENGTH = 20;
        private static final long serialVersionUID = MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT;


        /**
        * Timestamp in milliseconds since system boot
        */
        public int time_boot_ms;
        /**
        * Desired roll angle in radians
        */
        public float roll;
        /**
        * Desired pitch angle in radians
        */
        public float pitch;
        /**
        * Desired yaw angle in radians
        */
        public float yaw;
        /**
        * Collective thrust, normalized to 0 .. 1
        */
        public float thrust;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                packet.payload.putFloat(thrust);
                return packet;
        }

    /**
     * Decode a roll_pitch_yaw_thrust_setpoint message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
            thrust = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_roll_pitch_yaw_thrust_setpoint(){
        msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_roll_pitch_yaw_thrust_setpoint(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "ROLL_PITCH_YAW_THRUST_SETPOINT");
        //Log.d("MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT -"+" time_boot_ms:"+time_boot_ms+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+" thrust:"+thrust+"";
    }
}
```

```java
// MESSAGE SAFETY_ALLOWED_AREA PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Read out the safety zone the MAV currently assumes.
*/
public class msg_safety_allowed_area extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA = 55;
        public static final int MAVLINK_MSG_LENGTH = 25;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA;


        /**
        * x position 1 / Latitude 1
        */
        public float p1x;
        /**
        * y position 1 / Longitude 1
        */
        public float p1y;
        /**
        * z position 1 / Altitude 1
        */
        public float p1z;
        /**
        * x position 2 / Latitude 2
        */
        public float p2x;
        /**
        * y position 2 / Longitude 2
        */
        public float p2y;
        /**
        * z position 2 / Altitude 2
        */
        public float p2z;
        /**
        * Coordinate frame, as defined by MAV_FRAME enum in mavlink_types.h. Can be either global, GPS, right-handed with Z axis up or local, right handed, Z axis down.
        */
        public byte frame;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA;
                packet.payload.putFloat(p1x);
                packet.payload.putFloat(p1y);
                packet.payload.putFloat(p1z);
                packet.payload.putFloat(p2x);
                packet.payload.putFloat(p2y);
                packet.payload.putFloat(p2z);
                packet.payload.putByte(frame);
                return packet;
        }

    /**
     * Decode a safety_allowed_area message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            p1x = payload.getFloat();
            p1y = payload.getFloat();
            p1z = payload.getFloat();
            p2x = payload.getFloat();
            p2y = payload.getFloat();
            p2z = payload.getFloat();
            frame = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_safety_allowed_area(){
        msgid = MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_safety_allowed_area(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SAFETY_ALLOWED_AREA");
        //Log.d("MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA -"+" p1x:"+p1x+" p1y:"+p1y+" p1z:"+p1z+" p2x:"+p2x+" p2y:"+p2y+" p2z:"+p2z+" frame:"+frame+"";
    }
}
```

```java
// MESSAGE SAFETY_SET_ALLOWED_AREA PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set a safety zone (volume), which is defined by two corners of a cube. This message can be used to tell the MAV which setpoints/MISSIONs to accept and which to reject. Safety areas are often enforced by national or competition regulations.
*/
public class msg_safety_set_allowed_area extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA = 54;
        public static final int MAVLINK_MSG_LENGTH = 27;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;


        /**
        * x position 1 / Latitude 1
        */
        public float p1x;
        /**
        * y position 1 / Longitude 1
        */
        public float p1y;
        /**
        * z position 1 / Altitude 1
        */
        public float p1z;
        /**
        * x position 2 / Latitude 2
        */
        public float p2x;
        /**
        * y position 2 / Longitude 2
        */
        public float p2y;
        /**
        * z position 2 / Altitude 2
        */
        public float p2z;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * Coordinate frame, as defined by MAV_FRAME enum in mavlink_types.h. Can be either global, GPS, right-handed with Z axis up or local, right handed, Z axis down.
        */
        public byte frame;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;
                packet.payload.putFloat(p1x);
                packet.payload.putFloat(p1y);
                packet.payload.putFloat(p1z);
                packet.payload.putFloat(p2x);
                packet.payload.putFloat(p2y);
                packet.payload.putFloat(p2z);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(frame);
                return packet;
        }

    /**
     * Decode a safety_set_allowed_area message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            p1x = payload.getFloat();
            p1y = payload.getFloat();
            p1z = payload.getFloat();
            p2x = payload.getFloat();
            p2y = payload.getFloat();
            p2z = payload.getFloat();
            target_system = payload.getByte();
            target_component = payload.getByte();
            frame = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_safety_set_allowed_area(){
        msgid = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_safety_set_allowed_area(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SAFETY_SET_ALLOWED_AREA");
        //Log.d("MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA -"+" p1x:"+p1x+" p1y:"+p1y+" p1z:"+p1z+" p2x:"+p2x+" p2y:"+p2y+" p2z:"+p2z+" target_system:"+target_system+" target_component:"+target_component+" frame:"+frame+"";
    }
}
```

```java
// MESSAGE SCALED_IMU PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The RAW IMU readings for the usual 9DOF sensor setup. This message should contain the scaled values to the described units
*/
public class msg_scaled_imu extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SCALED_IMU = 26;
        public static final int MAVLINK_MSG_LENGTH = 22;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SCALED_IMU;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * X acceleration (mg)
        */
        public short xacc;
        /**
        * Y acceleration (mg)
        */
        public short yacc;
        /**
        * Z acceleration (mg)
        */
        public short zacc;
        /**
        * Angular speed around X axis (millirad /sec)
        */
        public short xgyro;
        /**
        * Angular speed around Y axis (millirad /sec)
        */
        public short ygyro;
        /**
        * Angular speed around Z axis (millirad /sec)
        */
        public short zgyro;
        /**
        * X Magnetic field (milli tesla)
        */
        public short xmag;
        /**
        * Y Magnetic field (milli tesla)
        */
        public short ymag;
        /**
        * Z Magnetic field (milli tesla)
        */
        public short zmag;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SCALED_IMU;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putShort(xacc);
                packet.payload.putShort(yacc);
                packet.payload.putShort(zacc);
                packet.payload.putShort(xgyro);
                packet.payload.putShort(ygyro);
                packet.payload.putShort(zgyro);
                packet.payload.putShort(xmag);
                packet.payload.putShort(ymag);
                packet.payload.putShort(zmag);
                return packet;
        }

    /**
     * Decode a scaled_imu message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            xacc = payload.getShort();
            yacc = payload.getShort();
            zacc = payload.getShort();
            xgyro = payload.getShort();
            ygyro = payload.getShort();
            zgyro = payload.getShort();
            xmag = payload.getShort();
            ymag = payload.getShort();
            zmag = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_scaled_imu(){
        msgid = MAVLINK_MSG_ID_SCALED_IMU;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_scaled_imu(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SCALED_IMU;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SCALED_IMU");
        //Log.d("MAVLINK_MSG_ID_SCALED_IMU", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SCALED_IMU -"+" time_boot_ms:"+time_boot_ms+" xacc:"+xacc+" yacc:"+yacc+" zacc:"+zacc+" xgyro:"+xgyro+" ygyro:"+ygyro+" zgyro:"+zgyro+" xmag:"+xmag+" ymag:"+ymag+" zmag:"+zmag+"";
    }
}
```

```java
// MESSAGE SCALED_PRESSURE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The pressure readings for the typical setup of one absolute and differential pressure sensor. The units are as specified in each field.
*/
public class msg_scaled_pressure extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SCALED_PRESSURE = 29;
        public static final int MAVLINK_MSG_LENGTH = 14;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SCALED_PRESSURE;


        /**
        * Timestamp (milliseconds since system boot)
        */
        public int time_boot_ms;
        /**
        * Absolute pressure (hectopascal)
        */
        public float press_abs;
        /**
        * Differential pressure 1 (hectopascal)
        */
        public float press_diff;
        /**
        * Temperature measurement (0.01 degrees celsius)
        */
        public short temperature;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SCALED_PRESSURE;
                packet.payload.putInt(time_boot_ms);
                packet.payload.putFloat(press_abs);
                packet.payload.putFloat(press_diff);
                packet.payload.putShort(temperature);
                return packet;
        }

    /**
     * Decode a scaled_pressure message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_boot_ms = payload.getInt();
            press_abs = payload.getFloat();
            press_diff = payload.getFloat();
            temperature = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_scaled_pressure(){
        msgid = MAVLINK_MSG_ID_SCALED_PRESSURE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_scaled_pressure(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SCALED_PRESSURE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SCALED_PRESSURE");
        //Log.d("MAVLINK_MSG_ID_SCALED_PRESSURE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SCALED_PRESSURE -"+" time_boot_ms:"+time_boot_ms+" press_abs:"+press_abs+" press_diff:"+press_diff+" temperature:"+temperature+"";
    }
}
```

```java
// MESSAGE SENSOR_OFFSETS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Offsets and calibrations values for hardware
        sensors. This makes it easier to debug the calibration process.
*/
public class msg_sensor_offsets extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SENSOR_OFFSETS = 150;
        public static final int MAVLINK_MSG_LENGTH = 42;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SENSOR_OFFSETS
;


        /**
        * magnetic declination (radians)
        */
        public float mag_declination;
        /**
        * raw pressure from barometer
        */
        public int raw_press;
        /**
        * raw temperature from barometer
        */
        public int raw_temp;
        /**
        * gyro X calibration
        */
        public float gyro_cal_x;
        /**
        * gyro Y calibration
        */
        public float gyro_cal_y;
        /**
        * gyro Z calibration
        */
        public float gyro_cal_z;
        /**
        * accel X calibration
        */
        public float accel_cal_x;
        /**
        * accel Y calibration
        */
        public float accel_cal_y;
        /**
        * accel Z calibration
        */
        public float accel_cal_z;
        /**
        * magnetometer X offset
        */
        public short mag_ofs_x;
        /**
        * magnetometer Y offset
        */
        public short mag_ofs_y;
        /**
        * magnetometer Z offset
        */
        public short mag_ofs_z;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SENSOR_OFFSETS;
                packet.payload.putFloat(mag_declination);
                packet.payload.putInt(raw_press);
                packet.payload.putInt(raw_temp);
                packet.payload.putFloat(gyro_cal_x);
                packet.payload.putFloat(gyro_cal_y);
                packet.payload.putFloat(gyro_cal_z);
                packet.payload.putFloat(accel_cal_x);
                packet.payload.putFloat(accel_cal_y);
                packet.payload.putFloat(accel_cal_z);
                packet.payload.putShort(mag_ofs_x);
                packet.payload.putShort(mag_ofs_y);
                packet.payload.putShort(mag_ofs_z);
                return packet;
        }

    /**
     * Decode a sensor_offsets message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            mag_declination = payload.getFloat();
            raw_press = payload.getInt();
            raw_temp = payload.getInt();
            gyro_cal_x = payload.getFloat();
            gyro_cal_y = payload.getFloat();
            gyro_cal_z = payload.getFloat();
            accel_cal_x = payload.getFloat();
            accel_cal_y = payload.getFloat();
            accel_cal_z = payload.getFloat();
            mag_ofs_x = payload.getShort();
            mag_ofs_y = payload.getShort();
            mag_ofs_z = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_sensor_offsets(){
        msgid = MAVLINK_MSG_ID_SENSOR_OFFSETS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_sensor_offsets(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SENSOR_OFFSETS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SENSOR_OFFSETS");
        //Log.d("MAVLINK_MSG_ID_SENSOR_OFFSETS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SENSOR_OFFSETS -"+" mag_declination:"+mag_declination+" raw_press:"+raw_press+" raw_temp:"+raw_temp+" gyro_cal_x:"+gyro_cal_x+" gyro_cal_y:"+gyro_cal_y+" gyro_cal_z:"+gyro_cal_z+" accel_cal_x:"+accel_cal_x+" accel_cal_y:"+accel_cal_y+" accel_cal_z:"+accel_cal_z+" mag_ofs_x:"+mag_ofs_x+" mag_ofs_y:"+mag_ofs_y+" mag_ofs_z:"+mag_ofs_z+"";
    }
}
```

```java
// MESSAGE SERVO_OUTPUT_RAW PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The RAW values of the servo outputs (for RC input from the remote, use the RC_CHANNELS messages). The standard PPM modulation is as follows: 1000 microseconds: 0%, 2000 microseconds: 100%.
*/
public class msg_servo_output_raw extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SERVO_OUTPUT_RAW = 36;
        public static final int MAVLINK_MSG_LENGTH = 21;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SERVO_OUTPUT_RAW;


        /**
        * Timestamp (microseconds since system boot)
        */
        public int time_usec;
        /**
        * Servo output 1 value, in microseconds
        */
        public short servo1_raw;
        /**
        * Servo output 2 value, in microseconds
        */
        public short servo2_raw;
        /**
        * Servo output 3 value, in microseconds
        */
        public short servo3_raw;
        /**
        * Servo output 4 value, in microseconds
        */
        public short servo4_raw;
        /**
        * Servo output 5 value, in microseconds
        */
        public short servo5_raw;
        /**
        * Servo output 6 value, in microseconds
        */
        public short servo6_raw;
        /**
        * Servo output 7 value, in microseconds
        */
        public short servo7_raw;
        /**
        * Servo output 8 value, in microseconds
        */
        public short servo8_raw;
        /**
        * Servo output port (set of 8 outputs = 1 port). Most MAVs will just use one, but this allows to encode more than 8 servos.
        */
        public byte port;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SERVO_OUTPUT_RAW;
                packet.payload.putInt(time_usec);
                packet.payload.putShort(servo1_raw);
                packet.payload.putShort(servo2_raw);
                packet.payload.putShort(servo3_raw);
                packet.payload.putShort(servo4_raw);
                packet.payload.putShort(servo5_raw);
                packet.payload.putShort(servo6_raw);
                packet.payload.putShort(servo7_raw);
                packet.payload.putShort(servo8_raw);
                packet.payload.putByte(port);
                return packet;
        }

    /**
     * Decode a servo_output_raw message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_usec = payload.getInt();
            servo1_raw = payload.getShort();
            servo2_raw = payload.getShort();
            servo3_raw = payload.getShort();
            servo4_raw = payload.getShort();
            servo5_raw = payload.getShort();
            servo6_raw = payload.getShort();
            servo7_raw = payload.getShort();
            servo8_raw = payload.getShort();
            port = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_servo_output_raw(){
        msgid = MAVLINK_MSG_ID_SERVO_OUTPUT_RAW;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_servo_output_raw(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SERVO_OUTPUT_RAW;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SERVO_OUTPUT_RAW");
        //Log.d("MAVLINK_MSG_ID_SERVO_OUTPUT_RAW", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SERVO_OUTPUT_RAW -"+" time_usec:"+time_usec+" servo1_raw:"+servo1_raw+" servo2_raw:"+servo2_raw+" servo3_raw:"+servo3_raw+" servo4_raw:"+servo4_raw+" servo5_raw:"+servo5_raw+" servo6_raw:"+servo6_raw+" servo7_raw:"+servo7_raw+" s
ervo8_raw:"+servo8_raw+" port:"+port+"";
    }
}
```

```java
// MESSAGE SETPOINT_6DOF PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set the 6 DOF setpoint for a attitude and position controller.
*/
public class msg_setpoint_6dof extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SETPOINT_6DOF = 149;
        public static final int MAVLINK_MSG_LENGTH = 25;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SETPOINT_6DOF;


        /**
        * Translational Component in x
        */
        public float trans_x;
        /**
        * Translational Component in y
        */
        public float trans_y;
        /**
        * Translational Component in z
        */
        public float trans_z;
        /**
        * Rotational Component in x
        */
        public float rot_x;
        /**
        * Rotational Component in y
        */
        public float rot_y;
        /**
        * Rotational Component in z
        */
        public float rot_z;
        /**
        * System ID
        */
        public byte target_system;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SETPOINT_6DOF;
                packet.payload.putFloat(trans_x);
                packet.payload.putFloat(trans_y);
                packet.payload.putFloat(trans_z);
                packet.payload.putFloat(rot_x);
                packet.payload.putFloat(rot_y);
                packet.payload.putFloat(rot_z);
                packet.payload.putByte(target_system);
                return packet;
        }

    /**
     * Decode a setpoint_6dof message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            trans_x = payload.getFloat();
            trans_y = payload.getFloat();
            trans_z = payload.getFloat();
            rot_x = payload.getFloat();
            rot_y = payload.getFloat();
            rot_z = payload.getFloat();
            target_system = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_setpoint_6dof(){
        msgid = MAVLINK_MSG_ID_SETPOINT_6DOF;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_setpoint_6dof(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SETPOINT_6DOF;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SETPOINT_6DOF");
        //Log.d("MAVLINK_MSG_ID_SETPOINT_6DOF", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SETPOINT_6DOF -"+" trans_x:"+trans_x+" trans_y:"+trans_y+" trans_z:"+trans_z+" rot_x:"+rot_x+" rot_y:"+rot_y+" rot_z:"+rot_z+" target_system:"+target_system+"";
    }
}
```

```java
// MESSAGE SETPOINT_8DOF PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set the 8 DOF setpoint for a controller.
*/
public class msg_setpoint_8dof extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SETPOINT_8DOF = 148;
        public static final int MAVLINK_MSG_LENGTH = 33;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SETPOINT_8DOF;


        /**
        * Value 1
        */
        public float val1;
        /**
        * Value 2
        */
        public float val2;
        /**
        * Value 3
        */
        public float val3;
        /**
        * Value 4
        */
        public float val4;
        /**
        * Value 5
        */
        public float val5;
        /**
        * Value 6
        */
        public float val6;
        /**
        * Value 7
        */
        public float val7;
        /**
        * Value 8
        */
        public float val8;
        /**
        * System ID
        */
        public byte target_system;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SETPOINT_8DOF;
                packet.payload.putFloat(val1);
                packet.payload.putFloat(val2);
                packet.payload.putFloat(val3);
                packet.payload.putFloat(val4);
                packet.payload.putFloat(val5);
                packet.payload.putFloat(val6);
                packet.payload.putFloat(val7);
                packet.payload.putFloat(val8);
                packet.payload.putByte(target_system);
                return packet;
        }

    /**
     * Decode a setpoint_8dof message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            val1 = payload.getFloat();
            val2 = payload.getFloat();
            val3 = payload.getFloat();
            val4 = payload.getFloat();
            val5 = payload.getFloat();
            val6 = payload.getFloat();
            val7 = payload.getFloat();
            val8 = payload.getFloat();
            target_system = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_setpoint_8dof(){
        msgid = MAVLINK_MSG_ID_SETPOINT_8DOF;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_setpoint_8dof(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SETPOINT_8DOF;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SETPOINT_8DOF");
        //Log.d("MAVLINK_MSG_ID_SETPOINT_8DOF", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SETPOINT_8DOF -"+" val1:"+val1+" val2:"+val2+" val3:"+val3+" val4:"+val4+" val5:"+val5+" val6:"+val6+" val7:"+val7+" val8:"+val8+" target_system:"+target_system+"";
    }
}
```

```java
// MESSAGE SET_GLOBAL_POSITION_SETPOINT_INT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set the current global position setpoint.
*/
public class msg_set_global_position_setpoint_int extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT = 53;
        public static final int MAVLINK_MSG_LENGTH = 15;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT;


        /**
        * WGS84 Latitude position in degrees * 1E7
        */
        public int latitude;
        /**
        * WGS84 Longitude position in degrees * 1E7
        */
        public int longitude;
        /**
        * WGS84 Altitude in meters * 1000 (positive for up)
        */
        public int altitude;
        /**
        * Desired yaw angle in degrees * 100
        */
        public short yaw;
        /**
        * Coordinate frame - valid values are only MAV_FRAME_GLOBAL or MAV_FRAME_GLOBAL_RELATIVE_ALT
        */
        public byte coordinate_frame;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT;
                packet.payload.putInt(latitude);
                packet.payload.putInt(longitude);
                packet.payload.putInt(altitude);
                packet.payload.putShort(yaw);
                packet.payload.putByte(coordinate_frame);
                return packet;
        }

    /**
     * Decode a set_global_position_setpoint_int message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            latitude = payload.getInt();
            longitude = payload.getInt();
            altitude = payload.getInt();
            yaw = payload.getShort();
            coordinate_frame = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_global_position_setpoint_int(){
        msgid = MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_global_position_setpoint_int(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_GLOBAL_POSITION_SETPOINT_INT");
        //Log.d("MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT -"+" latitude:"+latitude+" longitude:"+longitude+" altitude:"+altitude+" yaw:"+yaw+" coordinate_frame:"+coordinate_frame+"";
    }
}
```

```java
// MESSAGE SET_GPS_GLOBAL_ORIGIN PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* As local waypoints exist, the global MISSION reference allows to transform between the local coordinate frame and the global (GPS) coordinate frame. This can be necessary when e.g. in- and outdoor settings are connected and the MAV should move from in- to outdoor.
*/
public class msg_set_gps_global_origin extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN = 48;
        public static final int MAVLINK_MSG_LENGTH = 13;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;


        /**
        * global position * 1E7
        */
        public int latitude;
        /**
        * global position * 1E7
        */
        public int longitude;
        /**
        * global position * 1000
        */
        public int altitude;
        /**
        * System ID
        */
        public byte target_system;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;
                packet.payload.putInt(latitude);
                packet.payload.putInt(longitude);
                packet.payload.putInt(altitude);
                packet.payload.putByte(target_system);
                return packet;
        }

    /**
     * Decode a set_gps_global_origin message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            latitude = payload.getInt();
            longitude = payload.getInt();
            altitude = payload.getInt();
            target_system = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_gps_global_origin(){
        msgid = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_gps_global_origin(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_GPS_GLOBAL_ORIGIN");
        //Log.d("MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN -"+" latitude:"+latitude+" longitude:"+longitude+" altitude:"+altitude+" target_system:"+target_system+"";
    }
}
```

```java
// MESSAGE SET_LOCAL_POSITION_SETPOINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set the setpoint for a local position controller. This is the position in local coordinates the MAV should fly to. This message is sent by the path/MISSION planner to the onboard position controller. As some MAVs have a degree of freedom in yaw (e.g. all helicopters/quadrotors), the desired yaw angle is part of the message.
*/
public class msg_set_local_position_setpoint extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT = 50;
        public static final int MAVLINK_MSG_LENGTH = 19;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;


        /**
        * x position
        */
        public float x;
        /**
        * y position
        */
        public float y;
        /**
        * z position
        */
        public float z;
        /**
        * Desired yaw angle
        */
        public float yaw;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;
        /**
        * Coordinate frame - valid values are only MAV_FRAME_LOCAL_NED or MAV_FRAME_LOCAL_ENU
        */
        public byte coordinate_frame;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putFloat(yaw);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                packet.payload.putByte(coordinate_frame);
                return packet;
        }

    /**
     * Decode a set_local_position_setpoint message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            yaw = payload.getFloat();
            target_system = payload.getByte();
            target_component = payload.getByte();
            coordinate_frame = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_local_position_setpoint(){
        msgid = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_local_position_setpoint(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_LOCAL_POSITION_SETPOINT");
        //Log.d("MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT -"+" x:"+x+" y:"+y+" z:"+z+" yaw:"+yaw+" target_system:"+target_system+" target_component:"+target_component+" coordinate_frame:"+coordinate_frame+"";
    }
}
```

```java
// MESSAGE SET_MAG_OFFSETS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* set the magnetometer offsets
*/
public class msg_set_mag_offsets extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_MAG_OFFSETS = 151;
        public static final int MAVLINK_MSG_LENGTH = 8;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_MAG_OFFSETS;


        /**
        * magnetometer X offset
        */
        public short mag_ofs_x;
        /**
        * magnetometer Y offset
        */
        public short mag_ofs_y;
        /**
        * magnetometer Z offset
        */
        public short mag_ofs_z;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_MAG_OFFSETS;
                packet.payload.putShort(mag_ofs_x);
                packet.payload.putShort(mag_ofs_y);
                packet.payload.putShort(mag_ofs_z);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a set_mag_offsets message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            mag_ofs_x = payload.getShort();
            mag_ofs_y = payload.getShort();
            mag_ofs_z = payload.getShort();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_mag_offsets(){
        msgid = MAVLINK_MSG_ID_SET_MAG_OFFSETS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_mag_offsets(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_MAG_OFFSETS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_MAG_OFFSETS");
        //Log.d("MAVLINK_MSG_ID_SET_MAG_OFFSETS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_MAG_OFFSETS -"+" mag_ofs_x:"+mag_ofs_x+" mag_ofs_y:"+mag_ofs_y+" mag_ofs_z:"+mag_ofs_z+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE SET_QUAD_MOTORS_SETPOINT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Setpoint in the four motor speeds
*/
public class msg_set_quad_motors_setpoint extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT = 60;
        public static final int MAVLINK_MSG_LENGTH = 9;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT;


        /**
        * Front motor in + configuration, front left motor in x configuration
        */
        public short motor_front_nw;
        /**
        * Right motor in + configuration, front right motor in x configuration
        */
        public short motor_right_ne;
        /**
        * Back motor in + configuration, back right motor in x configuration
        */
        public short motor_back_se;
        /**
        * Left motor in + configuration, back left motor in x configuration
        */
        public short motor_left_sw;
        /**
        * System ID of the system that should set these motor commands
        */
        public byte target_system;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT;
                packet.payload.putShort(motor_front_nw);
                packet.payload.putShort(motor_right_ne);
                packet.payload.putShort(motor_back_se);
                packet.payload.putShort(motor_left_sw);
                packet.payload.putByte(target_system);
                return packet;
        }

    /**
     * Decode a set_quad_motors_setpoint message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            motor_front_nw = payload.getShort();
            motor_right_ne = payload.getShort();
            motor_back_se = payload.getShort();
            motor_left_sw = payload.getShort();
            target_system = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_quad_motors_setpoint(){
        msgid = MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_quad_motors_setpoint(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_QUAD_MOTORS_SETPOINT");
        //Log.d("MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT -"+" motor_front_nw:"+motor_front_nw+" motor_right_ne:"+motor_right_ne+" motor_back_se:"+motor_back_se+" motor_left_sw:"+motor_left_sw+" target_system:"+target_system+"";
    }
}
```

```java
// MESSAGE SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Setpoint for up to four quadrotors in a group / wing
*/
public class msg_set_quad_swarm_led_roll_pitch_yaw_thrust extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST = 63;
        public static final int MAVLINK_MSG_LENGTH = 46;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;


        /**
        * Desired roll angle in radians +-PI (+-32767)
        */
        public short roll[] = new short[4];
        /**
        * Desired pitch angle in radians +-PI (+-32767)
        */
        public short pitch[] = new short[4];
        /**
        * Desired yaw angle in radians, scaled to int16 +-PI (+-32767)
        */
        public short yaw[] = new short[4];
        /**
        * Collective thrust, scaled to uint16 (0..65535)
        */
        public short thrust[] = new short[4];
        /**
        * ID of the quadrotor group (0 - 255, up to 256 groups supported)
        */
        public byte group;
        /**
        * ID of the flight mode (0 - 255, up to 256 modes supported)
        */
        public byte mode;
        /**
        * RGB red channel (0-255)
        */
        public byte led_red[] = new byte[4];
        /**
        * RGB green channel (0-255)
        */
        public byte led_blue[] = new byte[4];
        /**
        * RGB blue channel (0-255)
        */
        public byte led_green[] = new byte[4];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;
                 for (int i = 0; i < roll.length; i++) {
                        packet.payload.putShort(roll[i]);
            }
                 for (int i = 0; i < pitch.length; i++) {
                        packet.payload.putShort(pitch[i]);
            }
                 for (int i = 0; i < yaw.length; i++) {
                        packet.payload.putShort(yaw[i]);
            }
                 for (int i = 0; i < thrust.length; i++) {
                        packet.payload.putShort(thrust[i]);
            }
                packet.payload.putByte(group);
                packet.payload.putByte(mode);
                 for (int i = 0; i < led_red.length; i++) {
                        packet.payload.putByte(led_red[i]);
            }
                 for (int i = 0; i < led_blue.length; i++) {
                        packet.payload.putByte(led_blue[i]);
            }
                 for (int i = 0; i < led_green.length; i++) {
                        packet.payload.putByte(led_green[i]);
            }
                return packet;
        }

    /**
     * Decode a set_quad_swarm_led_roll_pitch_yaw_thrust message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
             for (int i = 0; i < roll.length; i++) {
                        roll[i] = payload.getShort();
                }
             for (int i = 0; i < pitch.length; i++) {
                        pitch[i] = payload.getShort();
                }
             for (int i = 0; i < yaw.length; i++) {
                        yaw[i] = payload.getShort();
                }
             for (int i = 0; i < thrust.length; i++) {
                        thrust[i] = payload.getShort();
                }
            group = payload.getByte();
            mode = payload.getByte();
             for (int i = 0; i < led_red.length; i++) {
                        led_red[i] = payload.getByte();
                }
             for (int i = 0; i < led_blue.length; i++) {
                        led_blue[i] = payload.getByte();
                }
             for (int i = 0; i < led_green.length; i++) {
                        led_green[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_quad_swarm_led_roll_pitch_yaw_thrust(){
        msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_quad_swarm_led_roll_pitch_yaw_thrust(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST");
        //Log.d("MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST -"+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+" thrust:"+thrust+" group:"+group+" mode:"+mode+" led_red:"+led_red+" led_blue:"+led_blue+" led_green:"+led_green+"";
    }
}
```

```java
// MESSAGE SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Setpoint for up to four quadrotors in a group / wing
*/
public class msg_set_quad_swarm_roll_pitch_yaw_thrust extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST = 61;
        public static final int MAVLINK_MSG_LENGTH = 34;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST;


        /**
        * Desired roll angle in radians +-PI (+-32767)
        */
        public short roll[] = new short[4];
        /**
        * Desired pitch angle in radians +-PI (+-32767)
        */
        public short pitch[] = new short[4];
        /**
        * Desired yaw angle in radians, scaled to int16 +-PI (+-32767)
        */
        public short yaw[] = new short[4];
        /**
        * Collective thrust, scaled to uint16 (0..65535)
        */
        public short thrust[] = new short[4];
        /**
        * ID of the quadrotor group (0 - 255, up to 256 groups supported)
        */
        public byte group;
        /**
        * ID of the flight mode (0 - 255, up to 256 modes supported)
        */
        public byte mode;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST;
                 for (int i = 0; i < roll.length; i++) {
                        packet.payload.putShort(roll[i]);
            }
                 for (int i = 0; i < pitch.length; i++) {
                        packet.payload.putShort(pitch[i]);
            }
                 for (int i = 0; i < yaw.length; i++) {
                        packet.payload.putShort(yaw[i]);
            }
                 for (int i = 0; i < thrust.length; i++) {
                        packet.payload.putShort(thrust[i]);
            }
                packet.payload.putByte(group);
                packet.payload.putByte(mode);
                return packet;
    }

    /**
     * Decode a set_quad_swarm_roll_pitch_yaw_thrust message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
             for (int i = 0; i < roll.length; i++) {
                        roll[i] = payload.getShort();
                }
             for (int i = 0; i < pitch.length; i++) {
                        pitch[i] = payload.getShort();
                }
             for (int i = 0; i < yaw.length; i++) {
                        yaw[i] = payload.getShort();
                }
             for (int i = 0; i < thrust.length; i++) {
                        thrust[i] = payload.getShort();
                }
            group = payload.getByte();
            mode = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_quad_swarm_roll_pitch_yaw_thrust(){
        msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_quad_swarm_roll_pitch_yaw_thrust(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST");
        //Log.d("MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST -"+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+" thrust:"+thrust+" group:"+group+" mode:"+mode+"";
    }
}
```

```java
// MESSAGE SET_ROLL_PITCH_YAW_SPEED_THRUST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set roll, pitch and yaw.
*/
public class msg_set_roll_pitch_yaw_speed_thrust extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST = 57;
        public static final int MAVLINK_MSG_LENGTH = 18;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST;


        /**
        * Desired roll angular speed in rad/s
        */
        public float roll_speed;
        /**
        * Desired pitch angular speed in rad/s
        */
        public float pitch_speed;
        /**
        * Desired yaw angular speed in rad/s
        */
        public float yaw_speed;
        /**
        * Collective thrust, normalized to 0 .. 1
        */
        public float thrust;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST;
                packet.payload.putFloat(roll_speed);
                packet.payload.putFloat(pitch_speed);
                packet.payload.putFloat(yaw_speed);
                packet.payload.putFloat(thrust);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a set_roll_pitch_yaw_speed_thrust message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            roll_speed = payload.getFloat();
            pitch_speed = payload.getFloat();
            yaw_speed = payload.getFloat();
            thrust = payload.getFloat();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_roll_pitch_yaw_speed_thrust(){
        msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_roll_pitch_yaw_speed_thrust(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_ROLL_PITCH_YAW_SPEED_THRUST");
        //Log.d("MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST -"+" roll_speed:"+roll_speed+" pitch_speed:"+pitch_speed+" yaw_speed:"+yaw_speed+" thrust:"+thrust+" target_system:"+target_system+" target_component:"+target_component+"";
    }
}
```

```java
// MESSAGE SET_ROLL_PITCH_YAW_THRUST PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Set roll, pitch and yaw.
*/
public class msg_set_roll_pitch_yaw_thrust extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST = 56;
        public static final int MAVLINK_MSG_LENGTH = 18;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SET_ROLL_PITCH
_YAW_THRUST;


        /**
        * Desired roll angle in radians
        */
        public float roll;
        /**
        * Desired pitch angle in radians
        */
        public float pitch;
        /**
        * Desired yaw angle in radians
        */
        public float yaw;
        /**
        * Collective thrust, normalized to 0 .. 1
        */
        public float thrust;
        /**
        * System ID
        */
        public byte target_system;
        /**
        * Component ID
        */
        public byte target_component;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST;
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                packet.payload.putFloat(thrust);
                packet.payload.putByte(target_system);
                packet.payload.putByte(target_component);
                return packet;
        }

    /**
     * Decode a set_roll_pitch_yaw_thrust message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
            thrust = payload.getFloat();
            target_system = payload.getByte();
            target_component = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_set_roll_pitch_yaw_thrust(){
        msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_set_roll_pitch_yaw_thrust(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SET_ROLL_PITCH_YAW_THRUST");
        //Log.d("MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST -"+" roll:"+roll+" pitch:
"+pitch+" yaw:"+yaw+" thrust:"+thrust+" target_system:"+target_system+" target_component:
"+target_component+"";
    }
}
```

```java
// MESSAGE SIMSTATE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Status of simulation environment, if used
*/
public class msg_simstate extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SIMSTATE = 164;
        public static final int MAVLINK_MSG_LENGTH = 44;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SIMSTATE;


        /**
        * Roll angle (rad)
        */
        public float roll;
        /**
        * Pitch angle (rad)
        */
        public float pitch;
        /**
        * Yaw angle (rad)
        */
        public float yaw;
        /**
        * X acceleration m/s/s
        */
        public float xacc;
        /**
        * Y acceleration m/s/s
        */
        public float yacc;
        /**
        * Z acceleration m/s/s
        */
        public float zacc;
        /**
        * Angular speed around X axis rad/s
        */
        public float xgyro;
        /**
        * Angular speed around Y axis rad/s
        */
        public float ygyro;
        /**
        * Angular speed around Z axis rad/s
        */
        public float zgyro;
        /**
        * Latitude in degrees
        */
        public float lat;
        /**
        * Longitude in degrees
        */
        public float lng;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SIMSTATE;
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                packet.payload.putFloat(xacc);
                packet.payload.putFloat(yacc);
                packet.payload.putFloat(zacc);
                packet.payload.putFloat(xgyro);
                packet.payload.putFloat(ygyro);
                packet.payload.putFloat(zgyro);
                packet.payload.putFloat(lat);
                packet.payload.putFloat(lng);
                return packet;
        }

    /**
     * Decode a simstate message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
            xacc = payload.getFloat();
            yacc = payload.getFloat();
            zacc = payload.getFloat();
            xgyro = payload.getFloat();
            ygyro = payload.getFloat();
            zgyro = payload.getFloat();
            lat = payload.getFloat();
            lng = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_simstate(){
        msgid = MAVLINK_MSG_ID_SIMSTATE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_simstate(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SIMSTATE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SIMSTATE");
        //Log.d("MAVLINK_MSG_ID_SIMSTATE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SIMSTATE -"+" roll:"+roll+" pitch:"+pitch+" yaw:"+y
aw+" xacc:"+xacc+" yacc:"+yacc+" zacc:"+zacc+" xgyro:"+xgyro+" ygyro:"+ygyro+" zgyro:"+zg
yro+" lat:"+lat+" lng:"+lng+"";
    }
}
```

```java
// MESSAGE STATE_CORRECTION PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Corrects the systems state by adding an error correction term to the position and velocity, and by rotating the attitude by a correction angle.
*/
public class msg_state_correction extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_STATE_CORRECTION = 64;
        public static final int MAVLINK_MSG_LENGTH = 36;
        private static final long serialVersionUID = MAVLINK_MSG_ID_STATE_CORRECTION;


        /**
        * x position error
        */
        public float xErr;
        /**
        * y position error
        */
        public float yErr;
        /**
        * z position error
        */
        public float zErr;
        /**
        * roll error (radians)
        */
        public float rollErr;
        /**
        * pitch error (radians)
        */
        public float pitchErr;
        /**
        * yaw error (radians)
        */
        public float yawErr;
        /**
        * x velocity
        */
        public float vxErr;
        /**
        * y velocity
        */
        public float vyErr;
        /**
        * z velocity
        */
        public float vzErr;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
          */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_STATE_CORRECTION;
                packet.payload.putFloat(xErr);
                packet.payload.putFloat(yErr);
                packet.payload.putFloat(zErr);
                packet.payload.putFloat(rollErr);
                packet.payload.putFloat(pitchErr);
                packet.payload.putFloat(yawErr);
                packet.payload.putFloat(vxErr);
                packet.payload.putFloat(vyErr);
                packet.payload.putFloat(vzErr);
                return packet;
        }

    /**
     * Decode a state_correction message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            xErr = payload.getFloat();
            yErr = payload.getFloat();
            zErr = payload.getFloat();
            rollErr = payload.getFloat();
            pitchErr = payload.getFloat();
            yawErr = payload.getFloat();
            vxErr = payload.getFloat();
            vyErr = payload.getFloat();
            vzErr = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_state_correction(){
        msgid = MAVLINK_MSG_ID_STATE_CORRECTION;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_state_correction(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_STATE_CORRECTION;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "STATE_CORRECTION");
        //Log.d("MAVLINK_MSG_ID_STATE_CORRECTION", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_STATE_CORRECTION -"+" xErr:"+xErr+" yErr:"+yErr+" zErr:"+zErr+" rollErr:"+rollErr+" pitchErr:"+pitchErr+" yawErr:"+yawErr+" vxErr:"+vxErr+" vyErr:"+vyErr+" vzErr:"+vzErr+"";
    }
}
```

```java
// MESSAGE STATUSTEXT PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Status text message. These messages are printed in yellow in the COMM console of QGroundControl. WARNING: They consume quite some bandwidth, so use only for important status and error messages. If implemented wisely, these messages are buffered on the MCU and sent only at a limited rate (e.g. 10 Hz).
*/
public class msg_statustext extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_STATUSTEXT = 253;
        public static final int MAVLINK_MSG_LENGTH = 51;
        private static final long serialVersionUID = MAVLINK_MSG_ID_STATUSTEXT;


        /**
        * Severity of status. Relies on the definitions within RFC-5424. See enum MAV_SEVERITY.
        */
        public byte severity;
        /**
        * Status text message, without null termination character
        */
        public byte text[] = new byte[50];

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_STATUSTEXT;
                packet.payload.putByte(severity);
                 for (int i = 0; i < text.length; i++) {
                        packet.payload.putByte(text[i]);
            }
                return packet;
        }

    /**
     * Decode a statustext message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            severity = payload.getByte();
             for (int i = 0; i < text.length; i++) {
                        text[i] = payload.getByte();
                }
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_statustext(){
        msgid = MAVLINK_MSG_ID_STATUSTEXT;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_statustext(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_STATUSTEXT;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "STATUSTEXT");
        //Log.d("MAVLINK_MSG_ID_STATUSTEXT", toString());
    }

   /**
     * Sets the buffer of this message with a string, adds the necessary padding
     */
    public void setText(String str) {
      int len = Math.min(str.length(), 50);
      for (int i=0; i<len; i++) {
        text[i] = (byte) str.charAt(i);
      }
      for (int i=len; i<50; i++) {                      // padding for the rest of the buffer
        text[i] = 0;
      }
    }

    /**
         * Gets the message, formated as a string
         */
        public String getText() {
                String result = "";
                for (int i = 0; i < 50; i++) {
                        if (text[i] != 0)
                                result = result + (char) text[i];
                        else
                                break;
                }
                return result;

        }
    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_STATUSTEXT -"+" severity:"+severity+" text:"+text+"";
    }
}
```

```java
// MESSAGE SYSTEM_TIME PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The system time is the time of the master clock, typically the computer clock of the main onboard computer.
*/
public class msg_system_time extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SYSTEM_TIME = 2;
        public static final int MAVLINK_MSG_LENGTH = 12;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SYSTEM_TIME;


        /**
        * Timestamp of the master clock in microseconds since UNIX epoch.
        */
        public long time_unix_usec;
        /**
        * Timestamp of the component clock since boot time in milliseconds.
        */
        public int time_boot_ms;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SYSTEM_TIME;
                packet.payload.putLong(time_unix_usec);
                packet.payload.putInt(time_boot_ms);
                return packet;
        }

    /**
     * Decode a system_time message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            time_unix_usec = payload.getLong();
            time_boot_ms = payload.getInt();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_system_time(){
        msgid = MAVLINK_MSG_ID_SYSTEM_TIME;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_system_time(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SYSTEM_TIME;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SYSTEM_TIME");
        //Log.d("MAVLINK_MSG_ID_SYSTEM_TIME", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SYSTEM_TIME -"+" time_unix_usec:"+time_unix_usec+" time_boot_ms:"+time_boot_ms+"";
    }
}
```

```java
// MESSAGE SYS_STATUS PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* The general system state. If the system is following the MAVLink standard, the system state is mainly defined by three orthogonal states/modes: The system mode, which is either LOCKED (motors shut down and locked), MANUAL (system under RC control), GUIDED (system with autonomous position control, position setpoint controlled manually) or AUTO (system guided by path/waypoint planner). The NAV_MODE defined the current flight state: LIFTOFF (often an open-loop maneuver), LANDING, WAYPOINTS or VECTOR. This represents the internal navigation state machine. The system status shows wether the system is currently active or not and if an emergency occured. During the CRITICAL and EMERGENCY states the MAV is still considered to be active, but should start emergency procedures autonomously. After a failure occured it should first move from active to critical to allow manual intervention and then move to emergency after a certain timeout.
*/
public class msg_sys_status extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_SYS_STATUS = 1;
        public static final int MAVLINK_MSG_LENGTH = 31;
        private static final long serialVersionUID = MAVLINK_MSG_ID_SYS_STATUS;


        /**
        * Bitmask showing which onboard controllers and sensors are present. Value of 0: not present. Value of 1: present. Indices: 0: 3D gyro, 1: 3D acc, 2: 3D mag, 3: absolute pressure, 4: differential pressure, 5: GPS, 6: optical flow, 7: computer vision position, 8: laser based position, 9: external ground-truth (Vicon or Leica). Controllers: 10: 3D angular rate control 11: attitude stabilization, 12: yaw position, 13: z/altitude control, 14: x/y position control, 15: motor outputs / control
        */
        public int onboard_control_sensors_present;
        /**
        * Bitmask showing which onboard controllers and sensors are enabled:  Value of 0: not enabled. Value of 1: enabled. Indices: 0: 3D gyro, 1: 3D acc, 2: 3D mag, 3: absolute pressure, 4: differential pressure, 5: GPS, 6: optical flow, 7: computer vision position, 8: laser based position, 9: external ground-truth (Vicon or Leica). Controllers: 10: 3D angular rate control 11: attitude stabilization, 12: yaw position, 13: z/altitude control, 14: x/y position control, 15: motor outputs / control
        */
        public int onboard_control_sensors_enabled;
        /**
        * Bitmask showing which onboard controllers and sensors are operational or have an error:  Value of 0: not enabled. Value of 1: enabled. Indices: 0: 3D gyro, 1: 3D acc, 2: 3D mag, 3: absolute pressure, 4: differential pressure, 5: GPS, 6: optical flow, 7: computer vision position, 8: laser based position, 9: external ground-truth (Vicon or Leica). Controllers: 10: 3D angular rate control 11: attitude stabilization, 12: yaw position, 13: z/altitude control, 14: x/y position control, 15: motor outputs / control
        */
        public int onboard_control_sensors_health;
        /**
        * Maximum usage in percent of the mainloop time, (0%: 0, 100%: 1000) should be always below 1000
        */
        public short load;
        /**
        * Battery voltage, in millivolts (1 = 1 millivolt)
        */
        public short voltage_battery;
        /**
        * Battery current, in 10*milliamperes (1 = 10 milliampere), -1: autopilot does not measure the current
        */
        public short current_battery;
        /**
        * Communication drops in percent, (0%: 0, 100%: 10'000), (UART, I2C, SPI, CAN), dropped packets on all links (packets that were corrupted on reception on the MAV)
        */
        public short drop_rate_comm;
        /**
        * Communication errors (UART, I2C, SPI, CAN), dropped packets on all links (packets that were corrupted on reception on the MAV)
        */
        public short errors_comm;
        /**
        * Autopilot-specific errors
        */
        public short errors_count1;
        /**
        * Autopilot-specific errors
        */
        public short errors_count2;
        /**
        * Autopilot-specific errors
        */
        public short errors_count3;
        /**
        * Autopilot-specific errors
        */
        public short errors_count4;
        /**
        * Remaining battery energy: (0%: 0, 100%: 100), -1: autopilot estimate the remaining battery
        */
        public byte battery_remaining;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_SYS_STATUS;
                packet.payload.putInt(onboard_control_sensors_present);
                packet.payload.putInt(onboard_control_sensors_enabled);
                packet.payload.putInt(onboard_control_sensors_health);
                packet.payload.putShort(load);
                packet.payload.putShort(voltage_battery);
                packet.payload.putShort(current_battery);
                packet.payload.putShort(drop_rate_comm);
                packet.payload.putShort(errors_comm);
                packet.payload.putShort(errors_count1);
                packet.payload.putShort(errors_count2);
                packet.payload.putShort(errors_count3);
                packet.payload.putShort(errors_count4);
                packet.payload.putByte(battery_remaining);
                return packet;
        }

    /**
     * Decode a sys_status message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
```

```java
            onboard_control_sensors_present = payload.getInt();
            onboard_control_sensors_enabled = payload.getInt();
            onboard_control_sensors_health = payload.getInt();
            load = payload.getShort();
            voltage_battery = payload.getShort();
            current_battery = payload.getShort();
            drop_rate_comm = payload.getShort();
            errors_comm = payload.getShort();
            errors_count1 = payload.getShort();
            errors_count2 = payload.getShort();
            errors_count3 = payload.getShort();
            errors_count4 = payload.getShort();
            battery_remaining = payload.getByte();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_sys_status(){
        msgid = MAVLINK_MSG_ID_SYS_STATUS;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_sys_status(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_SYS_STATUS;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "SYS_STATUS");
        //Log.d("MAVLINK_MSG_ID_SYS_STATUS", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_SYS_STATUS -"+" onboard_control_sensors_present:"+o
nboard_control_sensors_present+" onboard_control_sensors_enabled:"+onboard_control_sensor
s_enabled+" onboard_control_sensors_health:"+onboard_control_sensors_health+" load:"+load
+" voltage_battery:"+voltage_battery+" current_battery:"+current_battery+" drop_rate_comm
:"+drop_rate_comm+" errors_comm:"+errors_comm+" errors_count1:"+errors_count1+" errors_co
unt2:"+errors_count2+" errors_count3:"+errors_count3+" errors_count4:"+errors_count4+" ba
ttery_remaining:"+battery_remaining+"";
    }
}
```

```java
// MESSAGE VFR_HUD PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Metrics typically displayed on a HUD for fixed wing aircraft
*/
public class msg_vfr_hud extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_VFR_HUD = 74;
        public static final int MAVLINK_MSG_LENGTH = 20;
        private static final long serialVersionUID = MAVLINK_MSG_ID_VFR_HUD;


        /**
        * Current airspeed in m/s
        */
        public float airspeed;
        /**
        * Current ground speed in m/s
        */
        public float groundspeed;
        /**
        * Current altitude (MSL), in meters
        */
        public float alt;
        /**
        * Current climb rate in meters/second
        */
        public float climb;
        /**
        * Current heading in degrees, in compass units (0..360, 0=north)
        */
        public short heading;
        /**
        * Current throttle setting in integer percent, 0 to 100
        */
        public short throttle;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_VFR_HUD;
                packet.payload.putFloat(airspeed);
                packet.payload.putFloat(groundspeed);
                packet.payload.putFloat(alt);
                packet.payload.putFloat(climb);
                packet.payload.putShort(heading);
                packet.payload.putShort(throttle);
                return packet;
        }

    /**
     * Decode a vfr_hud message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            airspeed = payload.getFloat();
            groundspeed = payload.getFloat();
            alt = payload.getFloat();
            climb = payload.getFloat();
            heading = payload.getShort();
            throttle = payload.getShort();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_vfr_hud(){
        msgid = MAVLINK_MSG_ID_VFR_HUD;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_vfr_hud(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_VFR_HUD;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "VFR_HUD");
        //Log.d("MAVLINK_MSG_ID_VFR_HUD", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_VFR_HUD -"+" airspeed:"+airspeed+" groundspeed:"+gr
oundspeed+" alt:"+alt+" climb:"+climb+" heading:"+heading+" throttle:"+throttle+"";
    }
}
```

```java
// MESSAGE VICON_POSITION_ESTIMATE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
*
*/
public class msg_vicon_position_estimate extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE = 104;
        public static final int MAVLINK_MSG_LENGTH = 32;
        private static final long serialVersionUID = MAVLINK_MSG_ID_VICON_POSITION
_ESTIMATE;


        /**
        * Timestamp (microseconds, synced to UNIX time or since system boot)
        */
        public long usec;
        /**
        * Global X position
        */
        public float x;
        /**
        * Global Y position
        */
        public float y;
        /**
        * Global Z position
        */
        public float z;
        /**
        * Roll angle in rad
        */
        public float roll;
        /**
        * Pitch angle in rad
        */
        public float pitch;
        /**
        * Yaw angle in rad
        */
        public float yaw;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE;
                packet.payload.putLong(usec);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                return packet;
        }

    /**
     * Decode a vicon_position_estimate message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            usec = payload.getLong();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_vicon_position_estimate(){
        msgid = MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_vicon_position_estimate(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "VICON_POSITION_ESTIMATE");
        //Log.d("MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE -"+" usec:"+usec+" x:"+x+"
 y:"+y+" z:"+z+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+"";
    }
}
```

```java
// MESSAGE VISION_POSITION_ESTIMATE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
*
*/
public class msg_vision_position_estimate extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE = 102;
        public static final int MAVLINK_MSG_LENGTH = 32;
        private static final long serialVersionUID = MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE;


        /**
        * Timestamp (microseconds, synced to UNIX time or since system boot)
        */
        public long usec;
        /**
        * Global X position
        */
        public float x;
        /**
        * Global Y position
        */
        public float y;
        /**
        * Global Z position
        */
        public float z;
        /**
        * Roll angle in rad
        */
        public float roll;
        /**
        * Pitch angle in rad
        */
        public float pitch;
        /**
        * Yaw angle in rad
        */
        public float yaw;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE;
                packet.payload.putLong(usec);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                packet.payload.putFloat(roll);
                packet.payload.putFloat(pitch);
                packet.payload.putFloat(yaw);
                return packet;
        }

    /**
     * Decode a vision_position_estimate message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            usec = payload.getLong();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
            roll = payload.getFloat();
            pitch = payload.getFloat();
            yaw = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_vision_position_estimate(){
        msgid = MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_vision_position_estimate(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "VISION_POSITION_ESTIMATE");
        //Log.d("MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE -"+" usec:"+usec+" x:"+x+" y:"+y+" z:"+z+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+"";
    }
}
```

```java
// MESSAGE VISION_SPEED_ESTIMATE PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
*
*/
public class msg_vision_speed_estimate extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE = 103;
        public static final int MAVLINK_MSG_LENGTH = 20;
        private static final long serialVersionUID = MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE;


        /**
        * Timestamp (microseconds, synced to UNIX time or since system boot)
        */
        public long usec;
        /**
        * Global X speed
        */
        public float x;
        /**
        * Global Y speed
        */
        public float y;
        /**
        * Global Z speed
        */
        public float z;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE;
                packet.payload.putLong(usec);
                packet.payload.putFloat(x);
                packet.payload.putFloat(y);
                packet.payload.putFloat(z);
                return packet;
        }

    /**
     * Decode a vision_speed_estimate message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            usec = payload.getLong();
            x = payload.getFloat();
            y = payload.getFloat();
            z = payload.getFloat();
    }

     /**
      * Constructor for a new message, just initializes the msgid
      */
    public msg_vision_speed_estimate(){
        msgid = MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_vision_speed_estimate(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "VISION_SPEED_ESTIMATE");
        //Log.d("MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE -"+" usec:"+usec+" x:"+x+" y:"+y+" z:"+z+"";
    }
}
```

```java
// MESSAGE WIND PACKING
package com.MAVLink.Messages.ardupilotmega;

import com.MAVLink.Messages.MAVLinkMessage;
import com.MAVLink.Messages.MAVLinkPayload;
import com.MAVLink.Messages.MAVLinkPacket;
//import android.util.Log;

/**
* Wind estimation
*/
public class msg_wind extends MAVLinkMessage{

        public static final int MAVLINK_MSG_ID_WIND = 168;
        public static final int MAVLINK_MSG_LENGTH = 12;
        private static final long serialVersionUID = MAVLINK_MSG_ID_WIND;


        /**
        * wind direction that wind is coming from (degrees)
        */
        public float direction;
        /**
        * wind speed in ground plane (m/s)
        */
        public float speed;
        /**
        * vertical wind speed (m/s)
        */
        public float speed_z;

        /**
         * Generates the payload for a mavlink message for a message of this type
         * @return
         */
        public MAVLinkPacket pack(){
                MAVLinkPacket packet = new MAVLinkPacket();
                packet.len = MAVLINK_MSG_LENGTH;
                packet.sysid = 255;
                packet.compid = 190;
                packet.msgid = MAVLINK_MSG_ID_WIND;
                packet.payload.putFloat(direction);
                packet.payload.putFloat(speed);
                packet.payload.putFloat(speed_z);
                return packet;
        }

    /**
     * Decode a wind message into this class fields
     *
     * @param payload The message to decode
     */
    public void unpack(MAVLinkPayload payload) {
        payload.resetIndex();
            direction = payload.getFloat();
            speed = payload.getFloat();
            speed_z = payload.getFloat();
    }

     /**
     * Constructor for a new message, just initializes the msgid
     */
    public msg_wind(){
        msgid = MAVLINK_MSG_ID_WIND;
    }

    /**
     * Constructor for a new message, initializes the message with the payload
     * from a mavlink packet
     *
     */
    public msg_wind(MAVLinkPacket mavLinkPacket){
        this.sysid = mavLinkPacket.sysid;
        this.compid = mavLinkPacket.compid;
        this.msgid = MAVLINK_MSG_ID_WIND;
        unpack(mavLinkPacket.payload);
        //Log.d("MAVLink", "WIND");
        //Log.d("MAVLINK_MSG_ID_WIND", toString());
    }


    /**
     * Returns a string with the MSG name and data
     */
    public String toString(){
        return "MAVLINK_MSG_ID_WIND -"+" direction:"+direction+" speed:"+speed+" s
peed_z:"+speed_z+"";
    }
}
```

```java
 package com.MAVLink.Messages;

/**
 * X.25 CRC calculation for MAVlink messages. The checksum must be initialized,
 * updated with witch field of the message, and then finished with the message
 * id.
 *
 */
public class CRC {
        private final int[] MAVLINK_MESSAGE_CRCS = {50, 124, 137, 0, 237, 217, 104, 119, 0, 0, 0, 89, 0, 0, 0, 0, 0, 0, 0, 0, 214, 159, 220, 168, 24, 23, 170, 144, 67, 115, 39, 246, 185, 104, 237, 244, 222, 212, 9, 254, 230, 28, 28, 132, 221, 232, 11, 153, 41, 39, 214, 223, 141, 33, 15, 3, 100, 24, 239, 238, 30, 240, 183, 130, 130, 0, 148, 21, 0, 243, 124, 0, 0, 0, 20, 0, 152, 143, 0, 0, 127, 106, 0, 0, 0, 0, 0, 0, 0, 231, 183, 63, 54, 0, 0, 0, 0, 0, 0, 0, 175, 102, 158, 208, 56, 93, 0, 0, 0, 0, 235, 93, 124, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 42, 241, 15, 134, 219, 208, 188, 84, 22, 19, 21, 134, 0, 78, 68, 189, 127, 111, 21, 21, 144, 1, 234, 73, 181, 22, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 204, 49, 170, 44, 83, 46, 0};
        private static final int CRC_INIT_VALUE = 0xffff;
        private int CRCvalue;

        /**
         * Accumulate the X.25 CRC by adding one char at a time.
         *
         * The checksum function adds the hash of one char at a time to the 16 bit
         * checksum (uint16_t).
         *
         * @param data
         *            new char to hash
         * @param crcAccum
         *            the already accumulated checksum
         **/
        public  void update_checksum(int data) {
                int tmp;
                data= data & 0xff;      //cast because we want an unsigned type
                tmp = data ^ (CRCvalue & 0xff);
                tmp ^= (tmp << 4) & 0xff;
                CRCvalue = ((CRCvalue >> 8) & 0xff) ^ (tmp << 8) ^ (tmp << 3)
                                ^ ((tmp >> 4) & 0xf);
        }

        /**
         * Finish the CRC calculation of a message, by running the CRC with the
         * Magic Byte. This Magic byte has been defined in MAVlink v1.0.
         *
         * @param msgid
         *            The message id number
         */
        public  void finish_checksum(int msgid) {
                update_checksum(MAVLINK_MESSAGE_CRCS[msgid]);
        }

        /**
         * Initialize the buffer for the X.25 CRC
         *
         */
        public void start_checksum() {
                CRCvalue = CRC_INIT_VALUE;
        }

        public int getMSB() {
                return ((CRCvalue >> 8) & 0xff);
        }

        public int getLSB() {
                return (CRCvalue & 0xff);
        }

        public CRC() {
                start_checksum();
        }

}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class FENCE_ACTION {
        public static final int FENCE_ACTION_NONE = 0; /* Disable fenced mode | */
        public static final int FENCE_ACTION_GUIDED = 1; /* Switched to guided mode to return point (fence point 0) | */
        public static final int FENCE_ACTION_REPORT = 2; /* Report fence breach, but don’t take action | */
        public static final int FENCE_ACTION_ENUM_END = 3; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class FENCE_BREACH {
        public static final int FENCE_BREACH_NONE = 0; /* No last fence breach | */
        public static final int FENCE_BREACH_MINALT = 1; /* Breached minimum altitude | */
        public static final int FENCE_BREACH_MAXALT = 2; /* Breached minimum altitude | */
        public static final int FENCE_BREACH_BOUNDARY = 3; /* Breached fence boundary | */
        public static final int FENCE_BREACH_ENUM_END = 4; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class LIMITS_STATE {
        public static final int LIMITS_INIT = 0; /*  pre-initialization | */
        public static final int LIMITS_DISABLED = 1; /*  disabled | */
        public static final int LIMITS_ENABLED = 2; /*  checking limits | */
        public static final int LIMITS_TRIGGERED = 3; /*  a limit has been breached | */
        public static final int LIMITS_RECOVERING = 4; /*  taking action eg. RTL | */
        public static final int LIMITS_RECOVERED = 5; /*  we’re no longer in breach of a limit | */
        public static final int LIMITS_STATE_ENUM_END = 6; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class LIMIT_MODULE {
        public static final int LIMIT_GPSLOCK = 1; /*  pre-initialization | */
        public static final int LIMIT_GEOFENCE = 2; /*  disabled | */
        public static final int LIMIT_ALTITUDE = 4; /*  checking limits | */
        public static final int LIMIT_MODULE_ENUM_END = 5; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class MAVLINK_DATA_STREAM_TYPE {
        public static final int MAVLINK_DATA_STREAM_IMG_JPEG = 1; /*  | */
        public static final int MAVLINK_DATA_STREAM_IMG_BMP = 2; /*  | */
        public static final int MAVLINK_DATA_STREAM_IMG_RAW8U = 3; /*  | */
        public static final int MAVLINK_DATA_STREAM_IMG_RAW32U = 4; /*  | */
        public static final int MAVLINK_DATA_STREAM_IMG_PGM = 5; /*  | */
        public static final int MAVLINK_DATA_STREAM_IMG_PNG = 6; /*  | */
        public static final int MAVLINK_DATA_STREAM_TYPE_ENUM_END = 7; /*  | */
}
```

```java
/** Micro air vehicle / autopilot classes. This identifies the individual model.
*/
package com.MAVLink.Messages.enums;

public class MAV_AUTOPILOT {
        public static final int MAV_AUTOPILOT_GENERIC = 0; /* Generic autopilot, full support for everything | */
        public static final int MAV_AUTOPILOT_PIXHAWK = 1; /* PIXHAWK autopilot, http://pixhawk.ethz.ch | */
        public static final int MAV_AUTOPILOT_SLUGS = 2; /* SLUGS autopilot, http://slugsuav.soe.ucsc.edu | */
        public static final int MAV_AUTOPILOT_ARDUPILOTMEGA = 3; /* ArduPilotMega / ArduCopter, http://diydrones.com | */
        public static final int MAV_AUTOPILOT_OPENPILOT = 4; /* OpenPilot, http://openpilot.org | */
        public static final int MAV_AUTOPILOT_GENERIC_WAYPOINTS_ONLY = 5; /* Generic autopilot only supporting simple waypoints | */
        public static final int MAV_AUTOPILOT_GENERIC_WAYPOINTS_AND_SIMPLE_NAVIGATION_ONLY = 6; /* Generic autopilot supporting waypoints and other simple navigation commands | */
        public static final int MAV_AUTOPILOT_GENERIC_MISSION_FULL = 7; /* Generic autopilot supporting the full mission command set | */
        public static final int MAV_AUTOPILOT_INVALID = 8; /* No valid autopilot, e.g. a GCS or other MAVLink component | */
        public static final int MAV_AUTOPILOT_PPZ = 9; /* PPZ UAV - http://nongnu.org/paparazzi | */
        public static final int MAV_AUTOPILOT_UDB = 10; /* UAV Dev Board | */
        public static final int MAV_AUTOPILOT_FP = 11; /* FlexiPilot | */
        public static final int MAV_AUTOPILOT_PX4 = 12; /* PX4 Autopilot - http://pixhawk.ethz.ch/px4/ | */
        public static final int MAV_AUTOPILOT_ENUM_END = 13; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class MAV_CMD {
        public static final int MAV_CMD_NAV_WAYPOINT = 16; /* Navigate to MISSION. |Hold time in decimal seconds. (ignored by fixed wing, time to stay at MISSION for rotary wing)| Acceptance radius in meters (if the sphere with this radius is hit, the MISSION counts as reached)| 0 to pass through the WP, if > 0 radius in meters to pass by WP. Positive value for clockwise orbit, negative value for counter-clockwise orbit. Allows trajectory control.| Desired yaw angle at MISSION (rotary wing)| Latitude| Longitude| Altitude|  */
        public static final int MAV_CMD_NAV_LOITER_UNLIM = 17; /* Loiter around this MISSION an unlimited amount of time |Empty| Empty| Radius around MISSION, in meters. If positive loiter clockwise, else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        public static final int MAV_CMD_NAV_LOITER_TURNS = 18; /* Loiter around this MISSION for X turns |Turns| Empty| Radius around MISSION, in meters. If positive loiter clockwise, else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        public static final int MAV_CMD_NAV_LOITER_TIME = 19; /* Loiter around this MISSION for X seconds |Seconds (decimal)| Empty| Radius around MISSION, in meters. If positive loiter clockwise, else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        public static final int MAV_CMD_NAV_RETURN_TO_LAUNCH = 20; /* Return to launch location |Empty| Empty| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_NAV_LAND = 21; /* Land at location |Empty| Empty| Empty| Desired yaw angle.| Latitude| Longitude| Altitude|  */
        public static final int MAV_CMD_NAV_TAKEOFF = 22; /* Takeoff from ground / hand |Minimum pitch (if airspeed sensor present), desired pitch without sensor| Empty| Empty| Yaw angle (if magnetometer present), ignored without magnetometer| Latitude| Longitude| Altitude|  */
        public static final int MAV_CMD_NAV_ROI = 80; /* Sets the region of interest (ROI) for a sensor set or the vehicle itself. This can then be used by the vehicles control system to control the vehicle attitude and the attitude of various sensors such as cameras. |Region of intereset mode. (see MAV_ROI enum)| MISSION index/ target ID. (see MAV_ROI enum)| ROI index (allows a vehicle to manage multiple ROI's)| Empty| x the location of the fixed ROI (see MAV_FRAME)| y| z|  */
        public static final int MAV_CMD_NAV_PATHPLANNING = 81; /* Control autonomous path planning on the MAV. |0: Disable local obstacle avoidance / local path planning (without resetting map), 1: Enable local path planning, 2: Enable and reset local path planning| 0: Disable full path planning (without resetting map), 1: Enable, 2: Enable and reset map/occupancy grid, 3: Enable and reset planned route, but not occupancy grid| Empty| Yaw angle at goal, in compass degrees, [0..360]| Latitude/X of goal| Longitude/Y of goal| Altitude/Z of goal|  */
        public static final int MAV_CMD_NAV_LAST = 95; /* NOP - This command is only used to mark the upper limit of the NAV/ACTION commands in the enumeration |Empty| Empty| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_CONDITION_DELAY = 112; /* Delay mission state machine. |Delay in seconds (decimal)| Empty| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_CONDITION_CHANGE_ALT = 113; /* Ascend/descend at rate.  Delay mission state machine until desired altitude reached. |Descent / Ascend rate (m/s)| Empty| Empty| Empty| Empty| Empty| Finish Altitude|  */
        public static final int MAV_CMD_CONDITION_DISTANCE = 114; /* Delay mission state machine until within desired distance of next NAV point. |Distance (meters)| Empty| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_CONDITION_YAW = 115; /* Reach a certain target angle. |target angle: [0-360], 0 is north| speed during yaw change:[deg per second]| direction: negative: counter clockwise, positive: clockwise [-1,1]| relative offset or absolute angle: [ 1,0]| Empty| Empty| Empty|  */
        public static final int MAV_CMD_CONDITION_LAST = 159; /* NOP - This command is only used to mark the upper limit of the CONDITION commands in the enumeration |Empty| Empty| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_SET_MODE = 176; /* Set system mode. |Mode, as defined by ENUM MAV_MODE| Empty| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_JUMP = 177; /* Jump to the desired command in the mission list.  Repeat this action only the specified number of times |Sequence number| Repeat count| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_CHANGE_SPEED = 178; /* Change speed and/or throttle set points. |Speed type (0=Airspeed, 1=Ground Speed)| Speed  (m/s, -1 indicates no change)| Throttle  ( Percent, -1 indicates no change)| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_SET_HOME = 179; /* Changes the home location either to the current location or a specified location. |Use current (1=use current location, 0=use specified location)| Empty| Empty| Empty| Latitude| Longitude| Altitude|  */
        public static final int MAV_CMD_DO_SET_PARAMETER = 180; /* Set a system parameter.  Caution!  Use of this command requires knowledge of the numeric enumeration value of the parameter. |Parameter number| Parameter value| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_SET_RELAY = 181; /* Set a relay to a condition. |Relay number| Setting (1=on, 0=off, others possible depending on system hardware)| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_REPEAT_RELAY = 182; /* Cycle a relay on and off for a desired number of cyles with a desired period. |Relay number| Cycle count| Cycle time (seconds, decimal)| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_SET_SERVO = 183; /* Set a servo to a desired PWM value. |Servo number| PWM (microseconds, 1000 to 2000 typical)| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_REPEAT_SERVO = 184; /* Cycle a between its nominal setting and a desired PWM for a desired number of cycles with a desired period. |Servo number| PWM (microseconds, 1000 to 2000 typical)| Cycle count| Cycle time (seconds)| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_CONTROL_VIDEO = 200; /* Control onboard camera system. |Camera ID (-1 for all)| Transmission: 0: disabled, 1: enabled compressed, 2: enabled raw| Transmission mode: 0: video stream, >0: single images every n seconds (decimal)| Recording: 0: disabled, 1: enabled compressed, 2: enabled raw| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_DIGICAM_CONFIGURE = 202; /* Mission command to configure an on-board camera controller system. |Modes: P, TV, AV, M, Etc| Shutter speed: Divisor number for one second| Aperture: F stop number| ISO number e.g. 80, 100, 200, Etc| Exposure type enumerator| Command Identity| Main engine cut-off time before camera trigger in seconds/10 (0 means no cut-off)|  */
        public static final int MAV_CMD_DO_DIGICAM_CONTROL = 203; /* Mission command to control an on-board camera controller system. |Session control e.g. show/hide lens| Zoom's absolute position| Zooming step value to offset zoom from the current position| Focus Locking, Unlocking or Re-locking| Shooting Command| Command Identity| Empty|  */
        public static final int MAV_CMD_DO_MOUNT_CONFIGURE = 204; /* Mission command to configure a camera or antenna mount |Mount operation mode (see MAV_MOUNT_MODE enum)| stabilize roll? (1 = yes, 0 = no)| stabilize pitch? (1 = yes, 0 = no)| stabilize yaw? (1 = yes, 0 = no)| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_MOUNT_CONTROL = 205; /* Mission command to control a camera or antenna mount |pitch(deg*100) or lat, depending on mount mode.| roll(deg*100) or lon depending on mount mode| yaw(deg*100) or alt (in cm) depending on mount mode| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_DO_LAST = 240; /* NOP - This command is only used to mark the upper limit of the DO commands in the enumeration |Empty| Empty| Empty| Empty| Empty| Empty| Empty|  */
        public static final int MAV_CMD_PREFLIGHT_CALIBRATION = 241; /* Trigger calibration. This command will be only accepted if in pre-flight mode. |Gyro calibration: 0: no, 1: yes| Magnetometer calibration: 0: no, 1: yes| Ground pressure: 0: no, 1: yes| Radio calibration: 0: no, 1: yes| Accelerometer calibration: 0: no, 1: yes| Empty| Empty|  */
        public static final int MAV_CMD_PREFLIGHT_SET_SENSOR_OFFSETS = 242; /* Set sensor offsets. This command will be only accepted if in pre-flight mode. |Sensor to adjust the offsets for: 0: gyros, 1: accelerometer, 2: magnetometer, 3: barometer, 4: optical flow| X axis offset (or generic dimension 1), in the sensor's raw units| Y axis offset (or generic dimension 2), in the sensor's raw units| Z axis offset (or generic dimension 3), in the sensor's raw units| Generic dimension 4, in the sensor's raw units| Generic dimension 5, in the sensor's raw units| Generic dimension 6, in the sensor's raw units|  */
        public static final int MAV_CMD_PREFLIGHT_STORAGE = 245; /* Request storage of different parameter values and logs. This command will be only accepted if in pre-flight mode. |Parameter storage: 0: READ FROM FLASH/EEPROM, 1: WRITE CURRENT TO FLASH/EEPROM| Mission storage: 0: READ FROM FLASH/EEPROM, 1: WRITE CURRENT TO FLASH/EEPROM| Reserved
| Reserved| Empty| Empty| Empty|  */
        public static final int MAV_CMD_PREFLIGHT_REBOOT_SHUTDOWN = 246; /* Reques
t the reboot or shutdown of system components. |0: Do nothing for autopilot, 1: Reboot au
topilot, 2: Shutdown autopilot.| 0: Do nothing for onboard computer, 1: Reboot onboard co
mputer, 2: Shutdown onboard computer.| Reserved| Reserved| Empty| Empty| Empty|  */
        public static final int MAV_CMD_OVERRIDE_GOTO = 252; /* Hold / continue th
e current action |MAV_GOTO_DO_HOLD: hold MAV_GOTO_DO_CONTINUE: continue with next item in
 mission plan| MAV_GOTO_HOLD_AT_CURRENT_POSITION: Hold at current position MAV_GOTO_HOLD_
AT_SPECIFIED_POSITION: hold at specified position| MAV_FRAME coordinate frame of hold poi
nt| Desired yaw angle in degrees| Latitude / X position| Longitude / Y position| Altitude
 / Z position|  */
        public static final int MAV_CMD_MISSION_START = 300; /* start running a mi
ssion |first_item: the first mission item to run| last_item:  the last mission item to ru
n (after this item is run, the mission ends)|  */
        public static final int MAV_CMD_COMPONENT_ARM_DISARM = 400; /* Arms / Disa
rms a component |1 to arm, 0 to disarm|  */
        public static final int MAV_CMD_ENUM_END = 401; /*  | */
}
```

```java
/** ACK / NACK / ERROR values as a result of MAV_CMDs and for mission item transmission.
*/
package com.MAVLink.Messages.enums;

public class MAV_CMD_ACK {
        public static final int MAV_CMD_ACK_OK = 1; /* Command / mission item is ok. | */
        public static final int MAV_CMD_ACK_ERR_FAIL = 2; /* Generic error message if none of the other reasons fails or if no detailed error reporting is implemented. | */
        public static final int MAV_CMD_ACK_ERR_ACCESS_DENIED = 3; /* The system is refusing to accept this command from this source / communication partner. | */
        public static final int MAV_CMD_ACK_ERR_NOT_SUPPORTED = 4; /* Command or mission item is not supported, other commands would be accepted. | */
        public static final int MAV_CMD_ACK_ERR_COORDINATE_FRAME_NOT_SUPPORTED = 5; /* The coordinate frame of this command / mission item is not supported. | */
        public static final int MAV_CMD_ACK_ERR_COORDINATES_OUT_OF_RANGE = 6; /* The coordinate frame of this command is ok, but he coordinate values exceed the safety limits of this system. This is a generic error, please use the more specific error messages below if possible. | */
        public static final int MAV_CMD_ACK_ERR_X_LAT_OUT_OF_RANGE = 7; /* The X or latitude value is out of range. | */
        public static final int MAV_CMD_ACK_ERR_Y_LON_OUT_OF_RANGE = 8; /* The Y or longitude value is out of range. | */
        public static final int MAV_CMD_ACK_ERR_Z_ALT_OUT_OF_RANGE = 9; /* The Z or altitude value is out of range. | */
        public static final int MAV_CMD_ACK_ENUM_END = 10; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class MAV_COMPONENT {
        public static final int MAV_COMP_ID_ALL = 0; /*  | */
        public static final int MAV_COMP_ID_CAMERA = 100; /*  | */
        public static final int MAV_COMP_ID_SERVO1 = 140; /*  | */
        public static final int MAV_COMP_ID_SERVO2 = 141; /*  | */
        public static final int MAV_COMP_ID_SERVO3 = 142; /*  | */
        public static final int MAV_COMP_ID_SERVO4 = 143; /*  | */
        public static final int MAV_COMP_ID_SERVO5 = 144; /*  | */
        public static final int MAV_COMP_ID_SERVO6 = 145; /*  | */
        public static final int MAV_COMP_ID_SERVO7 = 146; /*  | */
        public static final int MAV_COMP_ID_SERVO8 = 147; /*  | */
        public static final int MAV_COMP_ID_SERVO9 = 148; /*  | */
        public static final int MAV_COMP_ID_SERVO10 = 149; /*  | */
        public static final int MAV_COMP_ID_SERVO11 = 150; /*  | */
        public static final int MAV_COMP_ID_SERVO12 = 151; /*  | */
        public static final int MAV_COMP_ID_SERVO13 = 152; /*  | */
        public static final int MAV_COMP_ID_SERVO14 = 153; /*  | */
        public static final int MAV_COMP_ID_MAPPER = 180; /*  | */
        public static final int MAV_COMP_ID_MISSIONPLANNER = 190; /*  | */
        public static final int MAV_COMP_ID_PATHPLANNER = 195; /*  | */
        public static final int MAV_COMP_ID_IMU = 200; /*  | */
        public static final int MAV_COMP_ID_IMU_2 = 201; /*  | */
        public static final int MAV_COMP_ID_IMU_3 = 202; /*  | */
        public static final int MAV_COMP_ID_GPS = 220; /*  | */
        public static final int MAV_COMP_ID_UDP_BRIDGE = 240; /*  | */
        public static final int MAV_COMP_ID_UART_BRIDGE = 241; /*  | */
        public static final int MAV_COMP_ID_SYSTEM_CONTROL = 250; /*  | */
        public static final int MAV_COMPONENT_ENUM_END = 251; /*  | */
}
```

```
/** Data stream IDs. A data stream is not a fixed set of messages, but rather a
     recommendation to the autopilot software. Individual autopilots may or may not obey
     the recommended messages.
*/
package com.MAVLink.Messages.enums;

public class MAV_DATA_STREAM {
        public static final int MAV_DATA_STREAM_ALL = 0; /* Enable all data streams | */
        public static final int MAV_DATA_STREAM_RAW_SENSORS = 1; /* Enable IMU_RAW, GPS_RAW, GPS_STATUS packets. | */
        public static final int MAV_DATA_STREAM_EXTENDED_STATUS = 2; /* Enable GPS_STATUS, CONTROL_STATUS, AUX_STATUS | */
        public static final int MAV_DATA_STREAM_RC_CHANNELS = 3; /* Enable RC_CHANNELS_SCALED, RC_CHANNELS_RAW, SERVO_OUTPUT_RAW | */
        public static final int MAV_DATA_STREAM_RAW_CONTROLLER = 4; /* Enable ATTITUDE_CONTROLLER_OUTPUT, POSITION_CONTROLLER_OUTPUT, NAV_CONTROLLER_OUTPUT. | */
        public static final int MAV_DATA_STREAM_POSITION = 6; /* Enable LOCAL_POSITION, GLOBAL_POSITION/GLOBAL_POSITION_INT messages. | */
        public static final int MAV_DATA_STREAM_EXTRA1 = 10; /* Dependent on the autopilot | */
        public static final int MAV_DATA_STREAM_EXTRA2 = 11; /* Dependent on the autopilot | */
        public static final int MAV_DATA_STREAM_EXTRA3 = 12; /* Dependent on the autopilot | */
        public static final int MAV_DATA_STREAM_ENUM_END = 13; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class MAV_FRAME {
        public static final int MAV_FRAME_GLOBAL = 0; /* Global coordinate frame, WGS84 coordinate system. First value / x: latitude, second value / y: longitude, third value / z: positive altitude over mean sea level (MSL) | */
        public static final int MAV_FRAME_LOCAL_NED = 1; /* Local coordinate frame, Z-up (x: north, y: east, z: down). | */
        public static final int MAV_FRAME_MISSION = 2; /* NOT a coordinate frame, indicates a mission command. | */
        public static final int MAV_FRAME_GLOBAL_RELATIVE_ALT = 3; /* Global coordinate frame, WGS84 coordinate system, relative altitude over ground with respect to the home position. First value / x: latitude, second value / y: longitude, third value / z: positive altitude with 0 being at the altitude of the home location. | */
        public static final int MAV_FRAME_LOCAL_ENU = 4; /* Local coordinate frame, Z-down (x: east, y: north, z: up) | */
        public static final int MAV_FRAME_ENUM_END = 5; /*  | */
}
```

```java
/** Override command, pauses current mission execution and moves immediately to a position
*/
package com.MAVLink.Messages.enums;

public class MAV_GOTO {
        public static final int MAV_GOTO_DO_HOLD = 0; /* Hold at the current position. | */
        public static final int MAV_GOTO_DO_CONTINUE = 1; /* Continue with the next item in mission execution. | */
        public static final int MAV_GOTO_HOLD_AT_CURRENT_POSITION = 2; /* Hold at the current position of the system | */
        public static final int MAV_GOTO_HOLD_AT_SPECIFIED_POSITION = 3; /* Hold at the position specified in the parameters of the DO_HOLD action | */
        public static final int MAV_GOTO_ENUM_END = 4; /*  | */
}
```

```java
/** result in a mavlink mission ack
*/
package com.MAVLink.Messages.enums;

public class MAV_MISSION_RESULT {
        public static final int MAV_MISSION_ACCEPTED = 0; /* mission accepted OK | */
        public static final int MAV_MISSION_ERROR = 1; /* generic error / not accepting mission commands at all right now | */
        public static final int MAV_MISSION_UNSUPPORTED_FRAME = 2; /* coordinate frame is not supported | */
        public static final int MAV_MISSION_UNSUPPORTED = 3; /* command is not supported | */
        public static final int MAV_MISSION_NO_SPACE = 4; /* mission item exceeds storage space | */
        public static final int MAV_MISSION_INVALID = 5; /* one of the parameters has an invalid value | */
        public static final int MAV_MISSION_INVALID_PARAM1 = 6; /* param1 has an invalid value | */
        public static final int MAV_MISSION_INVALID_PARAM2 = 7; /* param2 has an invalid value | */
        public static final int MAV_MISSION_INVALID_PARAM3 = 8; /* param3 has an invalid value | */
        public static final int MAV_MISSION_INVALID_PARAM4 = 9; /* param4 has an invalid value | */
        public static final int MAV_MISSION_INVALID_PARAM5_X = 10; /* x/param5 has an invalid value | */
        public static final int MAV_MISSION_INVALID_PARAM6_Y = 11; /* y/param6 has an invalid value | */
        public static final int MAV_MISSION_INVALID_PARAM7 = 12; /* param7 has an invalid value | */
        public static final int MAV_MISSION_INVALID_SEQUENCE = 13; /* received waypoint out of sequence | */
        public static final int MAV_MISSION_DENIED = 14; /* not accepting any mission commands from this communication partner | */
        public static final int MAV_MISSION_RESULT_ENUM_END = 15; /*  | */
}
```

```java
/** These defines are predefined OR-combined mode flags. There is no need to use values from this enum, but it
                simplifies the use of the mode flags. Note that manual input is enabled in all modes as a safety override.
*/
package com.MAVLink.Messages.enums;

public class MAV_MODE {
        public static final int MAV_MODE_PREFLIGHT = 0; /* System is not ready to fly, booting, calibrating, etc. No flag is set. | */
        public static final int MAV_MODE_MANUAL_DISARMED = 64; /* System is allowed to be active, under manual (RC) control, no stabilization | */
        public static final int MAV_MODE_TEST_DISARMED = 66; /* UNDEFINED mode. This solely depends on the autopilot - use with caution, intended for developers only. | */
        public static final int MAV_MODE_STABILIZE_DISARMED = 80; /* System is allowed to be active, under assisted RC control. | */
        public static final int MAV_MODE_GUIDED_DISARMED = 88; /* System is allowed to be active, under autonomous control, manual setpoint | */
        public static final int MAV_MODE_AUTO_DISARMED = 92; /* System is allowed to be active, under autonomous control and navigation (the trajectory is decided onboard and not pre-programmed by MISSIONs) | */
        public static final int MAV_MODE_MANUAL_ARMED = 192; /* System is allowed to be active, under manual (RC) control, no stabilization | */
        public static final int MAV_MODE_TEST_ARMED = 194; /* UNDEFINED mode. This solely depends on the autopilot - use with caution, intended for developers only. | */
        public static final int MAV_MODE_STABILIZE_ARMED = 208; /* System is allowed to be active, under assisted RC control. | */
        public static final int MAV_MODE_GUIDED_ARMED = 216; /* System is allowed to be active, under autonomous control, manual setpoint | */
        public static final int MAV_MODE_AUTO_ARMED = 220; /* System is allowed to be active, under autonomous control and navigation (the trajectory is decided onboard and not pre-programmed by MISSIONs) | */
        public static final int MAV_MODE_ENUM_END = 221; /*  | */
}
```

```java
/** These flags encode the MAV mode.
*/
package com.MAVLink.Messages.enums;

public class MAV_MODE_FLAG {
        public static final int MAV_MODE_FLAG_CUSTOM_MODE_ENABLED = 1; /* 0b00000001 Reserved for future use. | */
        public static final int MAV_MODE_FLAG_TEST_ENABLED = 2; /* 0b00000010 system has a test mode enabled. This flag is intended for temporary system tests and should not be used for stable implementations. | */
        public static final int MAV_MODE_FLAG_AUTO_ENABLED = 4; /* 0b00000100 autonomous mode enabled, system finds its own goal positions. Guided flag can be set or not, depends on the actual implementation. | */
        public static final int MAV_MODE_FLAG_GUIDED_ENABLED = 8; /* 0b00001000 guided mode enabled, system flies MISSIONs / mission items. | */
        public static final int MAV_MODE_FLAG_STABILIZE_ENABLED = 16; /* 0b00010000 system stabilizes electronically its attitude (and optionally position). It needs however further control inputs to move around. | */
        public static final int MAV_MODE_FLAG_HIL_ENABLED = 32; /* 0b00100000 hardware in the loop simulation. All motors / actuators are blocked, but internal software is full operational. | */
        public static final int MAV_MODE_FLAG_MANUAL_INPUT_ENABLED = 64; /* 0b01000000 remote control input is enabled. | */
        public static final int MAV_MODE_FLAG_SAFETY_ARMED = 128; /* 0b10000000 MAV safety set to armed. Motors are enabled / running / can start. Ready to fly. | */
        public static final int MAV_MODE_FLAG_ENUM_END = 129; /*  | */
}
```

```java
/** These values encode the bit positions of the decode position. These values can be used to read the value of a flag bit by combining the base_mode variable with AND with the flag position value. The result will be either 0 or 1, depending on if the flag is set or not.
*/
package com.MAVLink.Messages.enums;

public class MAV_MODE_FLAG_DECODE_POSITION {
        public static final int MAV_MODE_FLAG_DECODE_POSITION_CUSTOM_MODE = 1; /* Eighth bit: 00000001 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_TEST = 2; /* Seventh bit: 00000010 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_AUTO = 4; /* Sixt bit:   00000100 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_GUIDED = 8; /* Fifth bit:  00001000 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_STABILIZE = 16; /* Fourth bit: 00010000 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_HIL = 32; /* Third bit:  00100000 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_MANUAL = 64; /* Second bit: 01000000 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_SAFETY = 128; /* First bit:  10000000 | */
        public static final int MAV_MODE_FLAG_DECODE_POSITION_ENUM_END = 129; /*  | */
}
```

```java
/** Enumeration of possible mount operation modes
*/
package com.MAVLink.Messages.enums;

public class MAV_MOUNT_MODE {
        public static final int MAV_MOUNT_MODE_RETRACT = 0; /* Load and keep safe position (Roll,Pitch,Yaw) from EEPROM and stop stabilization | */
        public static final int MAV_MOUNT_MODE_NEUTRAL = 1; /* Load and keep neutral position (Roll,Pitch,Yaw) from EEPROM. | */
        public static final int MAV_MOUNT_MODE_MAVLINK_TARGETING = 2; /* Load neutral position and start MAVLink Roll,Pitch,Yaw control with stabilization | */
        public static final int MAV_MOUNT_MODE_RC_TARGETING = 3; /* Load neutral position and start RC Roll,Pitch,Yaw control with stabilization | */
        public static final int MAV_MOUNT_MODE_GPS_POINT = 4; /* Load neutral position and start to point to Lat,Lon,Alt | */
        public static final int MAV_MOUNT_MODE_ENUM_END = 5; /*  | */
}
```

```java
/** Specifies the datatype of a MAVLink parameter.
*/
package com.MAVLink.Messages.enums;

public class MAV_PARAM_TYPE {
        public static final int MAV_PARAM_TYPE_UINT8 = 1; /* 8-bit unsigned integer | */
        public static final int MAV_PARAM_TYPE_INT8 = 2; /* 8-bit signed integer | */
        public static final int MAV_PARAM_TYPE_UINT16 = 3; /* 16-bit unsigned integer | */
        public static final int MAV_PARAM_TYPE_INT16 = 4; /* 16-bit signed integer | */
        public static final int MAV_PARAM_TYPE_UINT32 = 5; /* 32-bit unsigned integer | */
        public static final int MAV_PARAM_TYPE_INT32 = 6; /* 32-bit signed integer | */
        public static final int MAV_PARAM_TYPE_UINT64 = 7; /* 64-bit unsigned integer | */
        public static final int MAV_PARAM_TYPE_INT64 = 8; /* 64-bit signed integer | */
        public static final int MAV_PARAM_TYPE_REAL32 = 9; /* 32-bit floating-point | */
        public static final int MAV_PARAM_TYPE_REAL64 = 10; /* 64-bit floating-point | */
        public static final int MAV_PARAM_TYPE_ENUM_END = 11; /*  | */
}
```

```
/** result from a mavlink command
*/
package com.MAVLink.Messages.enums;

public class MAV_RESULT {
        public static final int MAV_RESULT_ACCEPTED = 0; /* Command ACCEPTED and E
XECUTED | */
        public static final int MAV_RESULT_TEMPORARILY_REJECTED = 1; /* Command TE
MPORARY REJECTED/DENIED | */
        public static final int MAV_RESULT_DENIED = 2; /* Command PERMANENTLY DENI
ED | */
        public static final int MAV_RESULT_UNSUPPORTED = 3; /* Command UNKNOWN/UNS
UPPORTED | */
        public static final int MAV_RESULT_FAILED = 4; /* Command executed, but fa
iled | */
        public static final int MAV_RESULT_ENUM_END = 5; /*  | */
}
```

```
/**  The ROI (region of interest) for the vehicle. This can be
                be used by the vehicle for camera/vehicle attitude alignment (see
                MAV_CMD_NAV_ROI).
*/
package com.MAVLink.Messages.enums;

public class MAV_ROI {
        public static final int MAV_ROI_NONE = 0; /* No region of interest. | */
        public static final int MAV_ROI_WPNEXT = 1; /* Point toward next MISSION.
| */
        public static final int MAV_ROI_WPINDEX = 2; /* Point toward given MISSION
. | */
        public static final int MAV_ROI_LOCATION = 3; /* Point toward fixed locati
on. | */
        public static final int MAV_ROI_TARGET = 4; /* Point toward of given id. |
 */
        public static final int MAV_ROI_ENUM_END = 5; /*  | */
}
```

```java
/** Indicates the severity level, generally used for status messages to indicate their relative urgency. Based on RFC-5424 using expanded definitions at: http://www.kiwisyslog.com/kb/info:-syslog-message-levels/.
*/
package com.MAVLink.Messages.enums;

public class MAV_SEVERITY {
	public static final int MAV_SEVERITY_EMERGENCY = 0; /* System is unusable. This is a "panic" condition. | */
	public static final int MAV_SEVERITY_ALERT = 1; /* Action should be taken immediately. Indicates error in non-critical systems. | */
	public static final int MAV_SEVERITY_CRITICAL = 2; /* Action must be taken immediately. Indicates failure in a primary system. | */
	public static final int MAV_SEVERITY_ERROR = 3; /* Indicates an error in secondary/redundant systems. | */
	public static final int MAV_SEVERITY_WARNING = 4; /* Indicates about a possible future error if this is not resolved within a given timeframe. Example would be a low battery warning. | */
	public static final int MAV_SEVERITY_NOTICE = 5; /* An unusual event has occured, though not an error condition. This should be investigated for the root cause. | */
	public static final int MAV_SEVERITY_INFO = 6; /* Normal operational messages. Useful for logging. No action is required for these messages. | */
	public static final int MAV_SEVERITY_DEBUG = 7; /* Useful non-operational messages that can assist in debugging. These should not occur during normal operation. | */
	public static final int MAV_SEVERITY_ENUM_END = 8; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class MAV_STATE {
        public static final int MAV_STATE_UNINIT = 0; /* Uninitialized system, state is unknown. | */
        public static final int MAV_STATE_BOOT = 1; /* System is booting up. | */
        public static final int MAV_STATE_CALIBRATING = 2; /* System is calibrating and not flight-ready. | */
        public static final int MAV_STATE_STANDBY = 3; /* System is grounded and on standby. It can be launched any time. | */
        public static final int MAV_STATE_ACTIVE = 4; /* System is active and might be already airborne. Motors are engaged. | */
        public static final int MAV_STATE_CRITICAL = 5; /* System is in a non-normal flight mode. It can however still navigate. | */
        public static final int MAV_STATE_EMERGENCY = 6; /* System is in a non-normal flight mode. It lost control over parts or over the whole airframe. It is in mayday and going down. | */
        public static final int MAV_STATE_POWEROFF = 7; /* System just initialized its power-down sequence, will shut down now. | */
        public static final int MAV_STATE_ENUM_END = 8; /*  | */
}
```

```java
/**
*/
package com.MAVLink.Messages.enums;

public class MAV_TYPE {
        public static final int MAV_TYPE_GENERIC = 0; /* Generic micro air vehicle. | */
        public static final int MAV_TYPE_FIXED_WING = 1; /* Fixed wing aircraft. | */
        public static final int MAV_TYPE_QUADROTOR = 2; /* Quadrotor | */
        public static final int MAV_TYPE_COAXIAL = 3; /* Coaxial helicopter | */
        public static final int MAV_TYPE_HELICOPTER = 4; /* Normal helicopter with tail rotor. | */
        public static final int MAV_TYPE_ANTENNA_TRACKER = 5; /* Ground installation | */
        public static final int MAV_TYPE_GCS = 6; /* Operator control unit / ground control station | */
        public static final int MAV_TYPE_AIRSHIP = 7; /* Airship, controlled | */
        public static final int MAV_TYPE_FREE_BALLOON = 8; /* Free balloon, uncontrolled | */
        public static final int MAV_TYPE_ROCKET = 9; /* Rocket | */
        public static final int MAV_TYPE_GROUND_ROVER = 10; /* Ground rover | */
        public static final int MAV_TYPE_SURFACE_BOAT = 11; /* Surface vessel, boat, ship | */
        public static final int MAV_TYPE_SUBMARINE = 12; /* Submarine | */
        public static final int MAV_TYPE_HEXAROTOR = 13; /* Hexarotor | */
        public static final int MAV_TYPE_OCTOROTOR = 14; /* Octorotor | */
        public static final int MAV_TYPE_TRICOPTER = 15; /* Octorotor | */
        public static final int MAV_TYPE_FLAPPING_WING = 16; /* Flapping wing | */
        public static final int MAV_TYPE_KITE = 17; /* Flapping wing | */
        public static final int MAV_TYPE_ENUM_END = 18; /*  | */
}
```

```

package com.MAVLink.Messages;

import java.io.Serializable;

public class MAVLinkMessage implements Serializable {
        private static final long serialVersionUID = -7754622750478538539L;
        // The MAVLink message classes have been changed to implement Serializable
,
        // this way is possible to pass a mavlink message trought the Service-Acct
ivity interface

        /**
         *  Simply a common interface for all MAVLink Messages
         */

        public  int sysid;
        public int compid;
        public int msgid;

}

```

```java
package com.MAVLink.Messages;

import android.util.Log;
import java.io.Serializable;
import com.MAVLink.Messages.ardupilotmega.*;

/**
 * Common interface for all MAVLink Messages
 * Packet Anatomy
 * This is the anatomy of one packet. It is inspired by the CAN and SAE AS-4 standards.

 * Byte Index  Content              Value       Explanation
 * 0            Packet start sign  v1.0: 0xFE   Indicates the start of a new packet.  (v0.9: 0x55)
 * 1            Payload length      0 - 255     Indicates length of the following payload.
 * 2            Packet sequence     0 - 255     Each component counts up his send sequence. Allows to detect packet loss
 * 3            System ID           1 - 255     ID of the SENDING system. Allows to differentiate different MAVs on the same network.
 * 4            Component ID        0 - 255     ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
 * 5            Message ID          0 - 255     ID of the message - the id defines what the payload means and how it should be correctly decoded.
 * 6 to (n+6)   Payload             0 - 255     Data of the message, depends on the message id.
 * (n+7)to(n+8) Checksum (low byte, high byte)  ITU X.25/SAE AS-4 hash, excluding packet start sign, so bytes 1..(n+6) Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).

 * The checksum is the same as used in ITU X.25 and SAE AS-4 standards (CRC-16-CCITT), documented in SAE AS5669A. Please see the MAVLink source code for a documented C-implementation of it. LINK TO CHECKSUM
 * The minimum packet length is 8 bytes for acknowledgement packets without payload
 * The maximum packet length is 263 bytes for full payload
 *
 * @author ghelle
 *
 */
public class MAVLinkPacket implements Serializable {
        private static final long serialVersionUID = 2095947771227815314L;

        public static final int MAVLINK_STX = 254;

        /**
         * Message length. NOT counting STX, LENGTH, SEQ, SYSID, COMPID, MSGID, CRC1 and CRC2
         */
        public int len;
        /**
         * Message sequence
         */
        public int seq;
        /**
         * ID of the SENDING system. Allows to differentiate different MAVs on the
         * same network.
         */
        public int sysid;
        /**
         * ID of the SENDING component. Allows to differentiate different components
         * of the same system, e.g. the IMU and the autopilot.
         */
        public int compid;
        /**
         * ID of the message - the id defines what the payload means and how it
         * should be correctly decoded.
         */
        public int msgid;
        /**
         * Data of the message, depends on the message id.
         */
        public MAVLinkPayload payload;
        /**
         * ITU X.25/SAE AS-4 hash, excluding packet start sign, so bytes 1..(n+6)
         * Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from
         * message fields. Protects the packet from decoding a different version of
         * the same packet but with different variables).
         */
        public CRC crc;

        public MAVLinkPacket(){
                payload = new MAVLinkPayload();
        }

        /**
         * Check if the size of the Payload is equal to the "len" byte
         */
        public boolean payloadIsFilled() {
                if (payload.size() >= MAVLinkPayload.MAX_PAYLOAD_SIZE-1) {
                        Log.d("MAV","Buffer overflow");
                        return true;
                }
                return (payload.size() == len);
        }

        /**
         * Update CRC for this packet.
         */
        public void generateCRC(){
                crc = new CRC();
                crc.update_checksum(len);
                crc.update_checksum(seq);
                crc.update_checksum(sysid);
                crc.update_checksum(compid);
                crc.update_checksum(msgid);
                payload.resetIndex();
                for (int i = 0; i < payload.size(); i++) {
                        crc.update_checksum(payload.getByte());
                }
                crc.finish_checksum(msgid);
    }

        /**
         * Encode this packet for transmission.
         *
         * @return Array with bytes to be transmitted
         */
        public byte[] encodePacket() {
                byte[] buffer = new byte[6 + len + 2];
                int i = 0;
                buffer[i++] = (byte) MAVLINK_STX;
                buffer[i++] = (byte) len;
                buffer[i++] = (byte) seq;
                buffer[i++] = (byte) sysid;
                buffer[i++] = (byte) compid;
                buffer[i++] = (byte) msgid;
                for (int j = 0; j < payload.size(); j++) {
                        buffer[i++] = payload.payload.get(j);
                }
                generateCRC();
                buffer[i++] = (byte) (crc.getLSB());
                buffer[i++] = (byte) (crc.getMSB());
                return buffer;
        }

        /**
         * Unpack the data in this packet and return a MAVLink message
         *
         * @return MAVLink message decoded from this packet
         */
        public MAVLinkMessage unpack() {
                switch (msgid) {
                case msg_sensor_offsets.MAVLINK_MSG_ID_SENSOR_OFFSETS:
                        return  new msg_sensor_offsets(this);
                case msg_set_mag_offsets.MAVLINK_MSG_ID_SET_MAG_OFFSETS:
                        return  new msg_set_mag_offsets(this);
                case msg_meminfo.MAVLINK_MSG_ID_MEMINFO:
                        return  new msg_meminfo(this);
                case msg_ap_adc.MAVLINK_MSG_ID_AP_ADC:
                        return  new msg_ap_adc(this);
                case msg_digicam_configure.MAVLINK_MSG_ID_DIGICAM_CONFIGURE:
                        return  new msg_digicam_configure(this);
                case msg_digicam_control.MAVLINK_MSG_ID_DIGICAM_CONTROL:
                        return  new msg_digicam_control(this);
                case msg_mount_configure.MAVLINK_MSG_ID_MOUNT_CONFIGURE:
                        return  new msg_mount_configure(this);
                case msg_mount_control.MAVLINK_MSG_ID_MOUNT_CONTROL:
                        return  new msg_mount_control(this);
                case msg_mount_status.MAVLINK_MSG_ID_MOUNT_STATUS:
                        return  new msg_mount_status(this);
                case msg_fence_point.MAVLINK_MSG_ID_FENCE_POINT:
                        return  new msg_fence_point(this);
                case msg_fence_fetch_point.MAVLINK_MSG_ID_FENCE_FETCH_POINT:
                        return  new msg_fence_fetch_point(this);
                case msg_fence_status.MAVLINK_MSG_ID_FENCE_STATUS:
                        return  new msg_fence_status(this);
                case msg_ahrs.MAVLINK_MSG_ID_AHRS:
                        return  new msg_ahrs(this);
                case msg_simstate.MAVLINK_MSG_ID_SIMSTATE:
                        return  new msg_simstate(this);
                case msg_hwstatus.MAVLINK_MSG_ID_HWSTATUS:
                        return  new msg_hwstatus(this);
                case msg_radio.MAVLINK_MSG_ID_RADIO:
                        return  new msg_radio(this);
                case msg_limits_status.MAVLINK_MSG_ID_LIMITS_STATUS:
                        return  new msg_limits_status(this);
                case msg_wind.MAVLINK_MSG_ID_WIND:
                        return  new msg_wind(this);
                case msg_data16.MAVLINK_MSG_ID_DATA16:
                        return  new msg_data16(this);
                case msg_data32.MAVLINK_MSG_ID_DATA32:
                        return  new msg_data32(this);
                case msg_data64.MAVLINK_MSG_ID_DATA64:
                        return  new msg_data64(this);
                case msg_data96.MAVLINK_MSG_ID_DATA96:
                        return  new msg_data96(this);
                case msg_heartbeat.MAVLINK_MSG_ID_HEARTBEAT:
                        return  new msg_heartbeat(this);
                case msg_sys_status.MAVLINK_MSG_ID_SYS_STATUS:
                        return  new msg_sys_status(this);
                case msg_system_time.MAVLINK_MSG_ID_SYSTEM_TIME:
                        return  new msg_system_time(this);
                case msg_ping.MAVLINK_MSG_ID_PING:
                        return  new msg_ping(this);
                case msg_change_operator_control.MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL:
                        return  new msg_change_operator_control(this);
                case msg_change_operator_control_ack.MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK:
                        return  new msg_change_operator_control_ack(this);
                case msg_auth_key.MAVLINK_MSG_ID_AUTH_KEY:
                        return  new msg_auth_key(this);
                case msg_set_mode.MAVLINK_MSG_ID_SET_MODE:
                        return  new msg_set_mode(this);
                case msg_param_request_read.MAVLINK_MSG_ID_PARAM_REQUEST_READ:
                        return  new msg_param_request_read(this);
                case msg_param_request_list.MAVLINK_MSG_ID_PARAM_REQUEST_LIST:
                        return  new msg_param_request_list(this);
                case msg_param_value.MAVLINK_MSG_ID_PARAM_VALUE:
                        return  new msg_param_value(this);
                case msg_param_set.MAVLINK_MSG_ID_PARAM_SET:
                        return  new msg_param_set(this);
                case msg_gps_raw_int.MAVLINK_MSG_ID_GPS_RAW_INT:
                        return  new msg_gps_raw_int(this);
                case msg_gps_status.MAVLINK_MSG_ID_GPS_STATUS:
                        return  new msg_gps_status(this);
                case msg_scaled_imu.MAVLINK_MSG_ID_SCALED_IMU:
                        return  new msg_scaled_imu(this);
                case msg_raw_imu.MAVLINK_MSG_ID_RAW_IMU:
                        return  new msg_raw_imu(this);
                case msg_raw_pressure.MAVLINK_MSG_ID_RAW_PRESSURE:
                        return  new msg_raw_pressure(this);
                case msg_scaled_pressure.MAVLINK_MSG_ID_SCALED_PRESSURE:
                        return  new msg_scaled_pressure(this);
                case msg_attitude.MAVLINK_MSG_ID_ATTITUDE:
                        return  new msg_attitude(this);
                case msg_attitude_quaternion.MAVLINK_MSG_ID_ATTITUDE_QUATERNION:
                        return  new msg_attitude_quaternion(this);
                case msg_local_position_ned.MAVLINK_MSG_ID_LOCAL_POSITION_NED:
                        return  new msg_local_position_ned(this);
                case msg_global_position_int.MAVLINK_MSG_ID_GLOBAL_POSITION_INT:
                        return  new msg_global_position_int(this);
                case msg_rc_channels_scaled.MAVLINK_MSG_ID_RC_CHANNELS_SCALED:
                        return  new msg_rc_channels_scaled(this);
                case msg_rc_channels_raw.MAVLINK_MSG_ID_RC_CHANNELS_RAW:
                        return  new msg_rc_channels_raw(this);
                case msg_servo_output_raw.MAVLINK_MSG_ID_SERVO_OUTPUT_RAW:
                        return  new msg_servo_output_raw(this);
                case msg_mission_request_partial_list.MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST:
                        return  new msg_mission_request_partial_list(this);
                case msg_mission_write_partial_list.MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST:
                        return  new msg_mission_write_partial_list(this);
                case msg_mission_item.MAVLINK_MSG_ID_MISSION_ITEM:
                        return  new msg_mission_item(this);
                case msg_mission_request.MAVLINK_MSG_ID_MISSION_REQUEST:
                        return  new msg_mission_request(this);
                case msg_mission_set_current.MAVLINK_MSG_ID_MISSION_SET_CURRENT:
                        return  new msg_mission_set_current(this);
                case msg_mission_current.MAVLINK_MSG_ID_MISSION_CURRENT:
                        return  new msg_mission_current(this);
                case msg_mission_request_list.MAVLINK_MSG_ID_MISSION_REQUEST_LIST:
                        return  new msg_mission_request_list(this);
                case msg_mission_count.MAVLINK_MSG_ID_MISSION_COUNT:
                        return  new msg_mission_count(this);
                case msg_mission_clear_all.MAVLINK_MSG_ID_MISSION_CLEAR_ALL:
                        return  new msg_mission_clear_all(this);
                case msg_mission_item_reached.MAVLINK_MSG_ID_MISSION_ITEM_REACHED:
                        return  new msg_mission_item_reached(this);
                case msg_mission_ack.MAVLINK_MSG_ID_MISSION_ACK:
```

```java
            return  new msg_mission_ack(this);
        case msg_set_gps_global_origin.MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN:
            return  new msg_set_gps_global_origin(this);
        case msg_gps_global_origin.MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN:
            return  new msg_gps_global_origin(this);
        case msg_set_local_position_setpoint.MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT:
            return  new msg_set_local_position_setpoint(this);
        case msg_local_position_setpoint.MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT:
            return  new msg_local_position_setpoint(this);
        case msg_global_position_setpoint_int.MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT:
            return  new msg_global_position_setpoint_int(this);
        case msg_set_global_position_setpoint_int.MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT:
            return  new msg_set_global_position_setpoint_int(this);
        case msg_safety_set_allowed_area.MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA:
            return  new msg_safety_set_allowed_area(this);
        case msg_safety_allowed_area.MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA:
            return new msg_safety_allowed_area(this);
        case msg_set_roll_pitch_yaw_thrust.MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST:
            return  new msg_set_roll_pitch_yaw_thrust(this);
        case msg_set_roll_pitch_yaw_speed_thrust.MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST:
            return  new msg_set_roll_pitch_yaw_speed_thrust(this);
        case msg_roll_pitch_yaw_thrust_setpoint.MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT:
            return  new msg_roll_pitch_yaw_thrust_setpoint(this);
        case msg_roll_pitch_yaw_speed_thrust_setpoint.MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT:
            return  new msg_roll_pitch_yaw_speed_thrust_setpoint(this);
        case msg_set_quad_motors_setpoint.MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT:
            return  new msg_set_quad_motors_setpoint(this);
        case msg_set_quad_swarm_roll_pitch_yaw_thrust.MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST:
            return  new msg_set_quad_swarm_roll_pitch_yaw_thrust(this);
        case msg_nav_controller_output.MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT:
            return  new msg_nav_controller_output(this);
        case msg_set_quad_swarm_led_roll_pitch_yaw_thrust.MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST:
            return  new msg_set_quad_swarm_led_roll_pitch_yaw_thrust(this);
        case msg_state_correction.MAVLINK_MSG_ID_STATE_CORRECTION:
            return new msg_state_correction(this);
        case msg_request_data_stream.MAVLINK_MSG_ID_REQUEST_DATA_STREAM:
            return new msg_request_data_stream(this);
        case msg_data_stream.MAVLINK_MSG_ID_DATA_STREAM:
            return new msg_data_stream(this);
        case msg_manual_control.MAVLINK_MSG_ID_MANUAL_CONTROL:
            return  new msg_manual_control(this);
        case msg_rc_channels_override.MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE:
            return new msg_rc_channels_override(this);
        case msg_vfr_hud.MAVLINK_MSG_ID_VFR_HUD:
            return  new msg_vfr_hud(this);
        case msg_command_long.MAVLINK_MSG_ID_COMMAND_LONG:
            return new msg_command_long(this);
        case msg_command_ack.MAVLINK_MSG_ID_COMMAND_ACK:
            return  new msg_command_ack(this);
        case msg_roll_pitch_yaw_rates_thrust_setpoint.MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT:
            return  new msg_roll_pitch_yaw_rates_thrust_setpoint(this);
        case msg_manual_setpoint.MAVLINK_MSG_ID_MANUAL_SETPOINT:
            return new msg_manual_setpoint(this);
        case msg_local_position_ned_system_global_offset.MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET:
            return  new msg_local_position_ned_system_global_offset(this);
        case msg_hil_state.MAVLINK_MSG_ID_HIL_STATE:
            return  new msg_hil_state(this);
        case msg_hil_controls.MAVLINK_MSG_ID_HIL_CONTROLS:
            return new msg_hil_controls(this);
        case msg_hil_rc_inputs_raw.MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW:
            return new msg_hil_rc_inputs_raw(this);
        case msg_optical_flow.MAVLINK_MSG_ID_OPTICAL_FLOW:
            return  new msg_optical_flow(this);
        case msg_global_vision_position_estimate.MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE:
            return  new msg_global_vision_position_estimate(this);
        case msg_vision_position_estimate.MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE:
            return  new msg_vision_position_estimate(this);
        case msg_vision_speed_estimate.MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE:
            return  new msg_vision_speed_estimate(this);
        case msg_vicon_position_estimate.MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE:
            return  new msg_vicon_position_estimate(this);
        case msg_highres_imu.MAVLINK_MSG_ID_HIGHRES_IMU:
            return  new msg_highres_imu(this);
        case msg_file_transfer_start.MAVLINK_MSG_ID_FILE_TRANSFER_START:
            return  new msg_file_transfer_start(this);
        case msg_file_transfer_dir_list.MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST:
            return  new msg_file_transfer_dir_list(this);
        case msg_file_transfer_res.MAVLINK_MSG_ID_FILE_TRANSFER_RES:
            return  new msg_file_transfer_res(this);
        case msg_battery_status.MAVLINK_MSG_ID_BATTERY_STATUS:
            return  new msg_battery_status(this);
        case msg_setpoint_8dof.MAVLINK_MSG_ID_SETPOINT_8DOF:
            return  new msg_setpoint_8dof(this);
        case msg_setpoint_6dof.MAVLINK_MSG_ID_SETPOINT_6DOF:
            return  new msg_setpoint_6dof(this);
        case msg_memory_vect.MAVLINK_MSG_ID_MEMORY_VECT:
            return  new msg_memory_vect(this);
        case msg_debug_vect.MAVLINK_MSG_ID_DEBUG_VECT:
            return  new msg_debug_vect(this);
        case msg_named_value_float.MAVLINK_MSG_ID_NAMED_VALUE_FLOAT:
            return  new msg_named_value_float(this);
        case msg_named_value_int.MAVLINK_MSG_ID_NAMED_VALUE_INT:
            return  new msg_named_value_int(this);
        case msg_statustext.MAVLINK_MSG_ID_STATUSTEXT:
            return  new msg_statustext(this);
        case msg_debug.MAVLINK_MSG_ID_DEBUG:
            return  new msg_debug(this);
        default:
            Log.d("MAVLink", "UNKNOW MESSAGE - " + msgid);
            return null;
        }
    }

}

```

```java
package com.MAVLink.Messages;

import java.nio.ByteBuffer;

public class MAVLinkPayload {

	public static final int MAX_PAYLOAD_SIZE = 512;

	public ByteBuffer payload;
	public int index;

	public MAVLinkPayload() {
		payload = ByteBuffer.allocate(MAX_PAYLOAD_SIZE);
	}

	public ByteBuffer getData() {
		return payload;
	}

	public int size() {
		return payload.position();
	}

	public void add(byte c) {
		payload.put(c);
	}

	public void resetIndex() {
		index = 0;
	}

	public byte getByte() {
		byte result = 0;
		result |= (payload.get(index + 0) & 0xFF);
		index += 1;
		return (byte) result;
	}

	public short getShort() {
		short result = 0;
		result |= (payload.get(index + 1) & 0xFF) << 8;
		result |= (payload.get(index + 0) & 0xFF);
		index += 2;
		return (short) result;
	}

	public int getInt() {
		int result = 0;
		result |= (payload.get(index + 3) & 0xFF) << 24;
		result |= (payload.get(index + 2) & 0xFF) << 16;
		result |= (payload.get(index + 1) & 0xFF) << 8;
		result |= (payload.get(index + 0) & 0xFF);
		index += 4;
		return (int) result;
	}

	public long getLong() {
		long result = 0;
		result |= (payload.get(index + 7) & 0xFF) << 56;
		result |= (payload.get(index + 6) & 0xFF) << 48;
		result |= (payload.get(index + 5) & 0xFF) << 40;
		result |= (payload.get(index + 4) & 0xFF) << 32;
		result |= (payload.get(index + 3) & 0xFF) << 24;
		result |= (payload.get(index + 2) & 0xFF) << 16;
		result |= (payload.get(index + 1) & 0xFF) << 8;
		result |= (payload.get(index + 0) & 0xFF);
		index += 8;
		return (long) result;
	}

	public float getFloat() {
		return Float.intBitsToFloat(getInt());
	}

	public void putByte(byte data) {
		add(data);
	}

	public void putShort(short data) {
		add((byte) (data >> 0));
		add((byte) (data >> 8));
	}

	public void putInt(int data) {
		add((byte) (data >> 0));
		add((byte) (data >> 8));
		add((byte) (data >> 16));
		add((byte) (data >> 24));
	}

	public void putLong(long data) {
		add((byte) (data >> 0));
		add((byte) (data >> 8));
		add((byte) (data >> 16));
		add((byte) (data >> 24));
		add((byte) (data >> 32));
		add((byte) (data >> 40));
		add((byte) (data >> 48));
		add((byte) (data >> 56));
	}

	public void putFloat(float data) {
		putInt(Float.floatToIntBits(data));
	}

}
```

```java
package com.MAVLink.Messages;


/**
 * Storage for MAVLink Packet and Error statistics
 * 
 * @author Helibot
 *
 */
public class MAVLinkStats /* implements Serializable */{

        public int receivedPacketCount;

        public int crcErrorCount;

        public int lostPacketCount;

        private int lastPacketSeq;

        /**
         * Check the new received packet to see if has lost someone between this and
         * the last packet
         * 
         * @param packet
         *            Packet that should be checked
         */
        public void newPacket(MAVLinkPacket packet) {
                advanceLastPacketSequence();
                if (hasLostPackets(packet)) {
                        updateLostPacketCount(packet);
                }
                lastPacketSeq = packet.seq;
                receivedPacketCount++;
        }

        private void updateLostPacketCount(MAVLinkPacket packet) {
                int lostPackets;
                if (packet.seq - lastPacketSeq < 0) {
                        lostPackets = (packet.seq - lastPacketSeq) + 255;
                } else {
                        lostPackets = (packet.seq - lastPacketSeq);
                }
                lostPacketCount += lostPackets;
        }

        private boolean hasLostPackets(MAVLinkPacket packet) {
                return lastPacketSeq > 0 && packet.seq != lastPacketSeq;
        }

        private void advanceLastPacketSequence() {
                // wrap from 255 to 0 if necessary
                lastPacketSeq = (lastPacketSeq + 1) & 0xFF;
        }

        /**
         * Called when a CRC error happens on the parser
         */
        public void crcError() {
                crcErrorCount++;
        }

        /**
         * Resets statistics for this MAVLink.
         */
        public void mavlinkResetStats() {
                lastPacketSeq = -1;
                lostPacketCount = 0;
                crcErrorCount = 0;
                receivedPacketCount = 0;
        }

}
```

```java
package com.MAVLink;

import com.MAVLink.Messages.MAVLinkPacket;
import com.MAVLink.Messages.MAVLinkStats;

public class Parser {

        /**
         * States from the parsing state machine
         */
        enum MAV_states {
                MAVLINK_PARSE_STATE_UNINIT, MAVLINK_PARSE_STATE_IDLE, MAVLINK_PARSE_STATE_GOT_STX, MAVLINK_PARSE_STATE_GOT_LENGTH, MAVLINK_PARSE_STATE_GOT_SEQ, MAVLINK_PARSE_STATE_GOT_SYSID, MAVLINK_PARSE_STATE_GOT_COMPID, MAVLINK_PARSE_STATE_GOT_MSGID, MAVLINK_PARSE_STATE_GOT_CRC1, MAVLINK_PARSE_STATE_GOT_PAYLOAD
        }

        MAV_states state = MAV_states.MAVLINK_PARSE_STATE_UNINIT;

        static boolean msg_received;

        public MAVLinkStats stats = new MAVLinkStats();
        private MAVLinkPacket m;

        /**
         * This is a convenience function which handles the complete MAVLink
         * parsing. the function will parse one byte at a time and return the
         * complete packet once it could be successfully decoded. Checksum and other
         * failures will be silently ignored.
         *
         * @param c
         *            The char to parse
         */
        public MAVLinkPacket mavlink_parse_char(int c) {
                msg_received = false;

                switch (state) {
                case MAVLINK_PARSE_STATE_UNINIT:
                case MAVLINK_PARSE_STATE_IDLE:

                        if (c == MAVLinkPacket.MAVLINK_STX) {
                                state = MAV_states.MAVLINK_PARSE_STATE_GOT_STX;
                                m = new MAVLinkPacket();
                        }
                        break;

                case MAVLINK_PARSE_STATE_GOT_STX:
                        if (msg_received) {
                                msg_received = false;
                                state = MAV_states.MAVLINK_PARSE_STATE_IDLE;
                        } else {
                                m.len = c;
                                state = MAV_states.MAVLINK_PARSE_STATE_GOT_LENGTH;
                        }
                        break;

                case MAVLINK_PARSE_STATE_GOT_LENGTH:
                        m.seq = c;
                        state = MAV_states.MAVLINK_PARSE_STATE_GOT_SEQ;
                        break;

                case MAVLINK_PARSE_STATE_GOT_SEQ:
                        m.sysid = c;
                        state = MAV_states.MAVLINK_PARSE_STATE_GOT_SYSID;
                        break;

                case MAVLINK_PARSE_STATE_GOT_SYSID:
                        m.compid = c;
                        state = MAV_states.MAVLINK_PARSE_STATE_GOT_COMPID;
                        break;

                case MAVLINK_PARSE_STATE_GOT_COMPID:
                        m.msgid = c;
                        if (m.len == 0) {
                                state = MAV_states.MAVLINK_PARSE_STATE_GOT_PAYLOAD;
                        } else {
                                state = MAV_states.MAVLINK_PARSE_STATE_GOT_MSGID;
                        }
                        break;

                case MAVLINK_PARSE_STATE_GOT_MSGID:
                        m.payload.add((byte) c);
                        if (m.payloadIsFilled()) {
                                state = MAV_states.MAVLINK_PARSE_STATE_GOT_PAYLOAD;
                        }
                        break;

                case MAVLINK_PARSE_STATE_GOT_PAYLOAD:
                        m.generateCRC();
                        // Check first checksum byte
                        if (c != m.crc.getLSB()) {
                                msg_received = false;
                                state = MAV_states.MAVLINK_PARSE_STATE_IDLE;
                                if (c == MAVLinkPacket.MAVLINK_STX) {
                                        state = MAV_states.MAVLINK_PARSE_STATE_GOT_STX;
                                        m.crc.start_checksum();
                                }
                                stats.crcError();
                        } else {
                                state = MAV_states.MAVLINK_PARSE_STATE_GOT_CRC1;
                        }
                        break;

                case MAVLINK_PARSE_STATE_GOT_CRC1:
                        // Check second checksum byte
                        if (c != m.crc.getMSB()) {
                                msg_received = false;
                                state = MAV_states.MAVLINK_PARSE_STATE_IDLE;
                                if (c == MAVLinkPacket.MAVLINK_STX) {
                                        state = MAV_states.MAVLINK_PARSE_STATE_GOT_STX;
                                        m.crc.start_checksum();
                                }
                                stats.crcError();
                        } else { // Successfully received the message
                                stats.newPacket(m);
                                msg_received = true;
                                state = MAV_states.MAVLINK_PARSE_STATE_IDLE;
                        }

                        break;

                }
                if (msg_received) {
                        return m;
                }else {
                        return null;
                }
        }



}
```