

```

1: package com.MAVLink.Messages;
2:
3: import java.util.ArrayList;
4:
5: import com.MAVLink.Messages.enums.MAV_CMD;
6:
7: public enum ApmCommands {
8:
9:     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 | */
10:    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 | */
11:    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 | */
12:    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 | */
13:    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 | */
14:    CMD_NAV_LAND("Land", MAV_CMD.MAV_CMD_NAV_LAND, CommandType.NAVIGATION), /* Land at location | Empty | Empty | Empty | Desired yaw angle. | Latitude | Longitude | Altitude | */
15:    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 | */
16:    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 interest 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 | */
17:    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 | */
18:    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 | */
19:    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 | */
20:    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 | */
21:    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 | */
22:    CMD_CONDITION_DISTANCE("Set Distance", MAV_CMD.MAV_CMD_CONDITION_DISTANCE, CommandType.COMMAND), /* Delay mission state machine until within desired distance of ne

```

```

xt NAV point. | Distance (meters) | Empty | Empty | Empty | Empty | Empty | Empty */
23:    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 no rth. | 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 | */
24:    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 | */
25:    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 cycles with a desired period. | Relay number | Cycle count | Cycle time (seconds, decimal) | Empty | Empty | Empty | Empty | Empty | */
26:    ;
27:
28:    private final String name;
29:    private final int arduPilotIntValue;
30:    private final CommandType commandType;
31:
32:    ApmCommands(String name, int type, CommandType showOnMap) {
33:        this.name = name;
34:        this.arduPilotIntValue = type;
35:        this.commandType = showOnMap;
36:    }
37:
38:    public String getName() {
39:        return name;
40:    }
41:
42:    public int getType() {
43:        return arduPilotIntValue;
44:    }
45:
46:    public boolean showOnMap() {
47:        switch (this.commandType) {
48:            case COMMAND:
49:                return false;
50:            default:
51:                case COMMAND_WITH_TARGET:
52:                case NAVIGATION:
53:                    return true;
54:        }
55:    }
56:
57:    public boolean isOnFlightPath() {
58:        switch (this.commandType) {
59:            default:
60:                case COMMAND:
61:                case COMMAND_WITH_TARGET:
62:                case NAVIGATION:
63:                    return true;
64:        }
65:    }
66:
67:
68:    public static ApmCommands getCmd(int type) {
69:        for (ApmCommands mode : ApmCommands.values()) {
70:            if (type == mode.getType()) {
71:                return mode;
72:            }
73:        }
74:        return null;
75:    }
76:
77:    public static ApmCommands getCmd(String str) {
78:        for (ApmCommands mode : ApmCommands.values()) {
79:            if (str.equals(mode.getName())) {

```

```
80:         return mode;
81:     }
82: }
83:     return null;
84: }
85:
86:     public static ArrayList<String> getNameList() {
87:         ArrayList<String> list = new ArrayList<String>();
88:
89:         for (ApmCommands mode : ApmCommands.values()) {
90:             list.add(mode.getName());
91:         }
92:         return list;
93:     }
94:
95:
96:     private enum CommandType{
97:         NAVIGATION,
98:         COMMAND,
99:         COMMAND_WITH_TARGET
100:     };
101:
102: }
```

```

1: package com.MAVLink.Messages;
2:
3: import java.util.ArrayList;
4: import java.util.List;
5:
6: import com.MAVLink.Messages.enums.MAV_TYPE;
7:
8: public enum ApmModes {
9:     FIXED_WING_MANUAL (0,"Manual",MAV_TYPE.MAV_TYPE_FIXED_WING),
10:    FIXED_WING_CIRCLE (1,"Circle",MAV_TYPE.MAV_TYPE_FIXED_WING),
11:    FIXED_WING_STABILIZE (2,"Stabilize",MAV_TYPE.MAV_TYPE_FIXED_WING),
12:    FIXED_WING_TRAINING (3,"Training",MAV_TYPE.MAV_TYPE_FIXED_WING),
13:    FIXED_WING_FLY_BY_WIRE_A (5,"FBW A",MAV_TYPE.MAV_TYPE_FIXED_WING),
14:    FIXED_WING_FLY_BY_WIRE_B (6,"FBW B",MAV_TYPE.MAV_TYPE_FIXED_WING),
15:    FIXED_WING_AUTO (10,"Auto",MAV_TYPE.MAV_TYPE_FIXED_WING),
16:    FIXED_WING_RTL (11,"RTL",MAV_TYPE.MAV_TYPE_FIXED_WING),
17:    FIXED_WING_LOITER (12,"Loiter",MAV_TYPE.MAV_TYPE_FIXED_WING),
18:    FIXED_WING_GUIDED (15,"Guided",MAV_TYPE.MAV_TYPE_FIXED_WING),
19:
20:    ROTOR_STABILIZE(0, "Stabilize", MAV_TYPE.MAV_TYPE_QUADROTOR),
21:    ROTOR_ACRO(1,"Acro", MAV_TYPE.MAV_TYPE_QUADROTOR),
22:    ROTOR_ALT_HOLD(2, "Alt Hold",MAV_TYPE.MAV_TYPE_QUADROTOR),
23:    ROTOR_AUTO(3, "Auto",MAV_TYPE.MAV_TYPE_QUADROTOR),
24:    ROTOR_GUIDED(4, "Guided",MAV_TYPE.MAV_TYPE_QUADROTOR),
25:    ROTOR_LOITER(5, "Loiter",MAV_TYPE.MAV_TYPE_QUADROTOR),
26:    ROTOR_RTL(6, "RTL",MAV_TYPE.MAV_TYPE_QUADROTOR),
27:    ROTOR_CIRCLE(7, "Circle",MAV_TYPE.MAV_TYPE_QUADROTOR),
28:    ROTOR_POSITION(8, "Pos Hold",MAV_TYPE.MAV_TYPE_QUADROTOR),
29:    ROTOR_LAND(9, "Land",MAV_TYPE.MAV_TYPE_QUADROTOR),
30:    ROTOR_TOY(11, "Toy",MAV_TYPE.MAV_TYPE_QUADROTOR),
31:    ROTOR_TAKEOFF(12, "Takeoff",MAV_TYPE.MAV_TYPE_QUADROTOR),
32:
33:    ROVER_MANUAL(0, "MANUAL", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
34:    ROVER_LEARNING(2, "LEARNING", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
35:    ROVER_STEERING(3, "STEERING", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
36:    ROVER_HOLD(4, "HOLD", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
37:    ROVER_AUTO(10, "AUTO", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
38:    ROVER_RTL(11, "RTL", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
39:    ROVER_GUIDED(15, "GUIDED", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
40:    ROVER_INITIALIZING(16, "INITIALIZING", MAV_TYPE.MAV_TYPE_GROUND_ROVER),
41:
42:
43:    UNKNOWN(-1, "Unknown",0);
44:
45:
46:
47:    private final int number;
48:    private final String name;
49:    private final int type;
50:
51:    ApmModes(int number,String name, int type){
52:        this.number = number;
53:        this.name = name;
54:        this.type = type;
55:    }
56:
57:    public int getNumber() {
58:        return number;
59:    }
60:
61:    public String getName() {
62:        return name;
63:    }
64:
65:    public int getType() {
66:        return type;
67:    }

```

```

68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121: }

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;
    }
}

```



```

1: // MESSAGE AHRS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * Status of DCM attitude estimator
11: */
12: public class msg_ahrs extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_AHRS = 163;
15:     public static final int MAVLINK_MSG_LENGTH = 28;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_AHRS;
17:
18:
19:     /**
20:     * X gyro drift estimate rad/s
21:     */
22:     public float omegaIx;
23:     /**
24:     * Y gyro drift estimate rad/s
25:     */
26:     public float omegaIy;
27:     /**
28:     * Z gyro drift estimate rad/s
29:     */
30:     public float omegaIz;
31:     /**
32:     * average accel_weight
33:     */
34:     public float accel_weight;
35:     /**
36:     * average renormalisation value
37:     */
38:     public float renorm_val;
39:     /**
40:     * average error_roll_pitch value
41:     */
42:     public float error_rp;
43:     /**
44:     * average error_yaw value
45:     */
46:     public float error_yaw;
47:
48:     /**
49:     * Generates the payload for a mavlink message for a message of this type
50:     * @return
51:     */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_AHRS;
58:         packet.payload.putFloat(omegaIx);
59:         packet.payload.putFloat(omegaIy);
60:         packet.payload.putFloat(omegaIz);
61:         packet.payload.putFloat(accel_weight);
62:         packet.payload.putFloat(renorm_val);
63:         packet.payload.putFloat(error_rp);
64:         packet.payload.putFloat(error_yaw);
65:         return packet;
66:     }
67:

```

```

68:     /**
69:     * Decode a ahrs message into this class fields
70:     *
71:     * @param payload The message to decode
72:     */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         omegaIx = payload.getFloat();
76:         omegaIy = payload.getFloat();
77:         omegaIz = payload.getFloat();
78:         accel_weight = payload.getFloat();
79:         renorm_val = payload.getFloat();
80:         error_rp = payload.getFloat();
81:         error_yaw = payload.getFloat();
82:     }
83:
84:     /**
85:     * Constructor for a new message, just initializes the msgid
86:     */
87:     public msg_ahrs(){
88:         msgid = MAVLINK_MSG_ID_AHRS;
89:     }
90:
91:     /**
92:     * Constructor for a new message, initializes the message with the payload
93:     * from a mavlink packet
94:     *
95:     */
96:     public msg_ahrs(MAVLinkPacket mavLinkPacket){
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compid = mavLinkPacket.compid;
99:         this.msgid = MAVLINK_MSG_ID_AHRS;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "AHRS");
102:         //Log.d("MAVLINK_MSG_ID_AHRS", toString());
103:     }
104:
105:
106:     /**
107:     * Returns a string with the MSG name and data
108:     */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_AHRS -"+" omegaIx:"+omegaIx+" omegaIy:"+omegaIy+" o
megaIz:"+omegaIz+" accel_weight:"+accel_weight+" renorm_val:"+renorm_val+" error_rp:"+err
or_rp+" error_yaw:"+error_yaw+"";
111:     }
112: }

```



```

1: // MESSAGE AP_ADC PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * raw ADC output
11:  */
12: public class msg_ap_adc extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_AP_ADC = 153;
15:     public static final int MAVLINK_MSG_LENGTH = 12;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_AP_ADC;
17:
18:
19:     /**
20:      * ADC output 1
21:      */
22:     public short adc1;
23:     /**
24:      * ADC output 2
25:      */
26:     public short adc2;
27:     /**
28:      * ADC output 3
29:      */
30:     public short adc3;
31:     /**
32:      * ADC output 4
33:      */
34:     public short adc4;
35:     /**
36:      * ADC output 5
37:      */
38:     public short adc5;
39:     /**
40:      * ADC output 6
41:      */
42:     public short adc6;
43:
44:     /**
45:      * Generates the payload for a mavlink message for a message of this type
46:      * @return
47:      */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_AP_ADC;
54:         packet.payload.putShort(adc1);
55:         packet.payload.putShort(adc2);
56:         packet.payload.putShort(adc3);
57:         packet.payload.putShort(adc4);
58:         packet.payload.putShort(adc5);
59:         packet.payload.putShort(adc6);
60:         return packet;
61:     }
62:
63:     /**
64:      * Decode a ap_adc message into this class fields
65:      *
66:      * @param payload The message to decode
67:      */

```

```

68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         adc1 = payload.getShort();
71:         adc2 = payload.getShort();
72:         adc3 = payload.getShort();
73:         adc4 = payload.getShort();
74:         adc5 = payload.getShort();
75:         adc6 = payload.getShort();
76:     }
77:
78:     /**
79:      * Constructor for a new message, just initializes the msgid
80:      */
81:     public msg_ap_adc(){
82:         msgid = MAVLINK_MSG_ID_AP_ADC;
83:     }
84:
85:     /**
86:      * Constructor for a new message, initializes the message with the payload
87:      * from a mavlink packet
88:      */
89:     public msg_ap_adc(MAVLinkPacket mavLinkPacket){
90:         this.sysid = mavLinkPacket.sysid;
91:         this.compid = mavLinkPacket.compid;
92:         this.msgid = MAVLINK_MSG_ID_AP_ADC;
93:         unpack(mavLinkPacket.payload);
94:         //Log.d("MAVLink", "AP_ADC");
95:         //Log.d("MAVLINK_MSG_ID_AP_ADC", toString());
96:     }
97:
98:
99:
100:     /**
101:      * Returns a string with the MSG name and data
102:      */
103:     public String toString(){
104:         return "MAVLINK_MSG_ID_AP_ADC -"+" adc1:"+adc1+" adc2:"+adc2+" adc3:"+adc3
105:         +" adc4:"+adc4+" adc5:"+adc5+" adc6:"+adc6+"";
106:     }

```



```

1: // MESSAGE ATTITUDE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The attitude in the aeronautical frame (right-handed, Z-down, X-front, Y-right).
11: */
12: public class msg_attitude extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_ATTITUDE = 30;
15:     public static final int MAVLINK_MSG_LENGTH = 28;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_ATTITUDE;
17:
18:
19:     /**
20:     * Timestamp (milliseconds since system boot)
21:     */
22:     public int time_boot_ms;
23:     /**
24:     * Roll angle (rad, -pi..+pi)
25:     */
26:     public float roll;
27:     /**
28:     * Pitch angle (rad, -pi..+pi)
29:     */
30:     public float pitch;
31:     /**
32:     * Yaw angle (rad, -pi..+pi)
33:     */
34:     public float yaw;
35:     /**
36:     * Roll angular speed (rad/s)
37:     */
38:     public float rollspeed;
39:     /**
40:     * Pitch angular speed (rad/s)
41:     */
42:     public float pitchspeed;
43:     /**
44:     * Yaw angular speed (rad/s)
45:     */
46:     public float yawspeed;
47:
48:     /**
49:     * Generates the payload for a mavlink message for a message of this type
50:     * @return
51:     */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_ATTITUDE;
58:         packet.payload.putInt(time_boot_ms);
59:         packet.payload.putFloat(roll);
60:         packet.payload.putFloat(pitch);
61:         packet.payload.putFloat(yaw);
62:         packet.payload.putFloat(rollspeed);
63:         packet.payload.putFloat(pitchspeed);
64:         packet.payload.putFloat(yawspeed);
65:         return packet;
66:     }
67:

```

```

68:     /**
69:     * Decode a attitude message into this class fields
70:     *
71:     * @param payload The message to decode
72:     */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         time_boot_ms = payload.getInt();
76:         roll = payload.getFloat();
77:         pitch = payload.getFloat();
78:         yaw = payload.getFloat();
79:         rollspeed = payload.getFloat();
80:         pitchspeed = payload.getFloat();
81:         yawspeed = payload.getFloat();
82:     }
83:
84:     /**
85:     * Constructor for a new message, just initializes the msgid
86:     */
87:     public msg_attitude(){
88:         msgid = MAVLINK_MSG_ID_ATTITUDE;
89:     }
90:
91:     /**
92:     * Constructor for a new message, initializes the message with the payload
93:     * from a mavlink packet
94:     *
95:     */
96:     public msg_attitude(MAVLinkPacket mavLinkPacket){
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compid = mavLinkPacket.compid;
99:         this.msgid = MAVLINK_MSG_ID_ATTITUDE;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "ATTITUDE");
102:         //Log.d("MAVLINK_MSG_ID_ATTITUDE", toString());
103:     }
104:
105:
106:     /**
107:     * Returns a string with the MSG name and data
108:     */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_ATTITUDE -"+" time_boot_ms:"+time_boot_ms+" roll:"+
roll+" pitch:"+pitch+" yaw:"+yaw+" rollspeed:"+rollspeed+" pitchspeed:"+pitchspeed+" yaws
peed:"+yawspeed+"";
111:     }
112: }

```



```

1: // MESSAGE ATTITUDE_QUATERNION PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The attitude in the aeronautical frame (right-handed, Z-down, X-front, Y-right),
    expressed as quaternion.
11: */
12: public class msg_attitude_quaternion extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_ATTITUDE_QUATERNION = 31;
15:     public static final int MAVLINK_MSG_LENGTH = 32;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_ATTITUDE_QUATE
RNION;
17:
18:     /**
19:     * Timestamp (milliseconds since system boot)
20:     */
21:     public int time_boot_ms;
22:     /**
23:     * Quaternion component 1
24:     */
25:     public float q1;
26:     /**
27:     * Quaternion component 2
28:     */
29:     public float q2;
30:     /**
31:     * Quaternion component 3
32:     */
33:     public float q3;
34:     /**
35:     * Quaternion component 4
36:     */
37:     public float q4;
38:     /**
39:     * Roll angular speed (rad/s)
40:     */
41:     public float rollspeed;
42:     /**
43:     * Pitch angular speed (rad/s)
44:     */
45:     public float pitchspeed;
46:     /**
47:     * Yaw angular speed (rad/s)
48:     */
49:     public float yawspeed;
50:
51:     /**
52:     * Generates the payload for a mavlink message for a message of this type
53:     * @return
54:     */
55:     public MAVLinkPacket pack(){
56:         MAVLinkPacket packet = new MAVLinkPacket();
57:         packet.len = MAVLINK_MSG_LENGTH;
58:         packet.sysid = 255;
59:         packet.compid = 190;
60:         packet.msgid = MAVLINK_MSG_ID_ATTITUDE_QUATERNION;
61:         packet.payload.putInt(time_boot_ms);
62:         packet.payload.putFloat(q1);
63:         packet.payload.putFloat(q2);
64:         packet.payload.putFloat(q3);
65:

```

```

66:         packet.payload.putFloat(q4);
67:         packet.payload.putFloat(rollspeed);
68:         packet.payload.putFloat(pitchspeed);
69:         packet.payload.putFloat(yawspeed);
70:         return packet;
71:     }
72:
73:     /**
74:     * Decode a attitude_quaternion message into this class fields
75:     *
76:     * @param payload The message to decode
77:     */
78:     public void unpack(MAVLinkPayload payload) {
79:         payload.resetIndex();
80:         time_boot_ms = payload.getInt();
81:         q1 = payload.getFloat();
82:         q2 = payload.getFloat();
83:         q3 = payload.getFloat();
84:         q4 = payload.getFloat();
85:         rollspeed = payload.getFloat();
86:         pitchspeed = payload.getFloat();
87:         yawspeed = payload.getFloat();
88:     }
89:
90:     /**
91:     * Constructor for a new message, just initializes the msgid
92:     */
93:     public msg_attitude_quaternion(){
94:         msgid = MAVLINK_MSG_ID_ATTITUDE_QUATERNION;
95:     }
96:
97:     /**
98:     * Constructor for a new message, initializes the message with the payload
99:     * from a mavlink packet
100:     */
101:
102:     public msg_attitude_quaternion(MAVLinkPacket mavLinkPacket){
103:         this.sysid = mavLinkPacket.sysid;
104:         this.compid = mavLinkPacket.compid;
105:         this.msgid = MAVLINK_MSG_ID_ATTITUDE_QUATERNION;
106:         unpack(mavLinkPacket.payload);
107:         //Log.d("MAVLink", "ATTITUDE_QUATERNION");
108:         //Log.d("MAVLINK_MSG_ID_ATTITUDE_QUATERNION", toString());
109:     }
110:
111:
112:     /**
113:     * Returns a string with the MSG name and data
114:     */
115:     public String toString(){
116:         return "MAVLINK_MSG_ID_ATTITUDE_QUATERNION -"+" time_boot_ms:"+time_boot_m
s+" q1:"+q1+" q2:"+q2+" q3:"+q3+" q4:"+q4+" rollspeed:"+rollspeed+" pitchspe
ed+" yawspeed:"+yawspeed+"";
117:     }
118: }

```



```

1: // MESSAGE AUTH_KEY PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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.
11:  */
12: public class msg_auth_key extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_AUTH_KEY = 7;
15:     public static final int MAVLINK_MSG_LENGTH = 32;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_AUTH_KEY;
17:
18:     /**
19:      * key
20:      */
21:     public byte key[] = new byte[32];
22:
23:     /**
24:      * Generates the payload for a mavlink message for a message of this type
25:      * @return
26:      */
27:     public MAVLinkPacket pack(){
28:         MAVLinkPacket packet = new MAVLinkPacket();
29:         packet.len = MAVLINK_MSG_LENGTH;
30:         packet.sysid = 255;
31:         packet.compid = 190;
32:         packet.msgid = MAVLINK_MSG_ID_AUTH_KEY;
33:         for (int i = 0; i < key.length; i++) {
34:             packet.payload.putByte(key[i]);
35:         }
36:         return packet;
37:     }
38:
39:     /**
40:      * Decode a auth_key message into this class fields
41:      *
42:      * @param payload The message to decode
43:      */
44:     public void unpack(MAVLinkPayload payload) {
45:         payload.resetIndex();
46:         for (int i = 0; i < key.length; i++) {
47:             key[i] = payload.getByte();
48:         }
49:     }
50:
51:     /**
52:      * Constructor for a new message, just initializes the msgid
53:      */
54:     public msg_auth_key(){
55:         msgid = MAVLINK_MSG_ID_AUTH_KEY;
56:     }
57:
58:     /**
59:      * Constructor for a new message, initializes the message with the payload
60:      * from a mavlink packet
61:      */
62:     public msg_auth_key(MAVLinkPacket mavLinkPacket){
63:         this.sysid = mavLinkPacket.sysid;
64:         this.compid = mavLinkPacket.compid;
65:         this.msgid = MAVLINK_MSG_ID_AUTH_KEY;
66:         unpack(mavLinkPacket.payload);
67:         //Log.d("MAVLink", "AUTH_KEY");
68:         //Log.d("MAVLINK_MSG_ID_AUTH_KEY", toString());
69:     }
70:
71:     /**
72:      * Sets the buffer of this message with a string, adds the necessary padding
73:      */
74:     public void setKey(String str) {
75:         int len = Math.min(str.length(), 32);
76:         for (int i=0; i<len; i++) {
77:             key[i] = (byte) str.charAt(i);
78:         }
79:         for (int i=len; i<32; i++) {
80:             key[i] = 0;
81:         }
82:     }
83:
84:     /**
85:      * Gets the message, formatted as a string
86:      */
87:     public String getKey() {
88:         String result = "";
89:         for (int i = 0; i < 32; i++) {
90:             if (key[i] != 0)
91:                 result = result + (char) key[i];
92:             else
93:                 break;
94:         }
95:         return result;
96:     }
97:
98:     /**
99:      * Returns a string with the MSG name and data
100:      */
101:     public String toString(){
102:         return "MAVLINK_MSG_ID_AUTH_KEY -" + key + key.length;
103:     }
104:
105: }
106:

```



```

1: // MESSAGE BATTERY_STATUS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Transmits battery informations for a accu pack.
11:  */
12: public class msg_battery_status extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_BATTERY_STATUS = 147;
15:     public static final int MAVLINK_MSG_LENGTH = 16;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_BATTERY_STATUS
;
17:
18:
19:     /**
20:      * Battery voltage of cell 1, in millivolts (1 = 1 millivolt)
21:      */
22:     public short voltage_cell_1;
23:     /**
24:      * Battery voltage of cell 2, in millivolts (1 = 1 millivolt), -1: no cell
25:      */
26:     public short voltage_cell_2;
27:     /**
28:      * Battery voltage of cell 3, in millivolts (1 = 1 millivolt), -1: no cell
29:      */
30:     public short voltage_cell_3;
31:     /**
32:      * Battery voltage of cell 4, in millivolts (1 = 1 millivolt), -1: no cell
33:      */
34:     public short voltage_cell_4;
35:     /**
36:      * Battery voltage of cell 5, in millivolts (1 = 1 millivolt), -1: no cell
37:      */
38:     public short voltage_cell_5;
39:     /**
40:      * Battery voltage of cell 6, in millivolts (1 = 1 millivolt), -1: no cell
41:      */
42:     public short voltage_cell_6;
43:     /**
44:      * Battery current, in 10*milliamperes (1 = 10 milliampere), -1: autopilot
does not measure the current
45:      */
46:     public short current_battery;
47:     /**
48:      * Accupack ID
49:      */
50:     public byte accu_id;
51:     /**
52:      * Remaining battery energy: (0%: 0, 100%: 100), -1: autopilot does not estimate the remaining battery
53:      */
54:     public byte battery_remaining;
55:
56:     /**
57:      * Generates the payload for a mavlink message for a message of this type
58:      * @return
59:      */
60:     public MAVLinkPacket pack(){
61:         MAVLinkPacket packet = new MAVLinkPacket();
62:         packet.len = MAVLINK_MSG_LENGTH;
63:         packet.sysid = 255;
64:         packet.compuid = 190;

```

```

65:         packet.msgid = MAVLINK_MSG_ID_BATTERY_STATUS;
66:         packet.payload.putShort(voltage_cell_1);
67:         packet.payload.putShort(voltage_cell_2);
68:         packet.payload.putShort(voltage_cell_3);
69:         packet.payload.putShort(voltage_cell_4);
70:         packet.payload.putShort(voltage_cell_5);
71:         packet.payload.putShort(voltage_cell_6);
72:         packet.payload.putShort(current_battery);
73:         packet.payload.putByte(accu_id);
74:         packet.payload.putByte(battery_remaining);
75:         return packet;
76:     }
77:
78:     /**
79:      * Decode a battery_status message into this class fields
80:      *
81:      * @param payload The message to decode
82:      */
83:     public void unpack(MAVLinkPayload payload) {
84:         payload.resetIndex();
85:         voltage_cell_1 = payload.getShort();
86:         voltage_cell_2 = payload.getShort();
87:         voltage_cell_3 = payload.getShort();
88:         voltage_cell_4 = payload.getShort();
89:         voltage_cell_5 = payload.getShort();
90:         voltage_cell_6 = payload.getShort();
91:         current_battery = payload.getShort();
92:         accu_id = payload.getByte();
93:         battery_remaining = payload.getByte();
94:     }
95:
96:     /**
97:      * Constructor for a new message, just initializes the msgid
98:      */
99:     public msg_battery_status(){
100:         msgid = MAVLINK_MSG_ID_BATTERY_STATUS;
101:     }
102:
103:     /**
104:      * Constructor for a new message, initializes the message with the payload
105:      * from a mavlink packet
106:      */
107:     public msg_battery_status(MAVLinkPacket mavLinkPacket){
108:         this.sysid = mavLinkPacket.sysid;
109:         this.compuid = mavLinkPacket.compuid;
110:         this.msgid = MAVLINK_MSG_ID_BATTERY_STATUS;
111:         unpack(mavLinkPacket.payload);
112:         //Log.d("MAVLink", "BATTERY_STATUS");
113:         //Log.d("MAVLINK_MSG_ID_BATTERY_STATUS", toString());
114:     }
115:
116:
117:
118:     /**
119:      * Returns a string with the MSG name and data
120:      */
121:     public String toString(){
122:         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+"";
123:     }
124: }

```



```

1: // MESSAGE CHANGE_OPERATOR_CONTROL PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Request to control this MAV
11:  */
12: public class msg_change_operator_control extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL = 5;
15:     public static final int MAVLINK_MSG_LENGTH = 28;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;
17:
18:
19:     /**
20:      * System the GCS requests control for
21:      */
22:     public byte target_system;
23:     /**
24:      * 0: request control of this MAV, 1: Release control of this MAV
25:      */
26:     public byte control_request;
27:     /**
28:      * 0: key as plaintext, 1-255: future, different hashing/encryption variant
29:      */
30:     public byte version;
31:     /**
32:      * Password / Key, depending on version plaintext or encrypted. 25 or less
33:      * characters, NULL terminated. The characters may involve A-Z, a-z, 0-9, and "!,.-"
34:      */
35:     public byte passkey[] = new byte[25];
36:
37:     /**
38:      * Generates the payload for a mavlink message for a message of this type
39:      * @return
40:      */
41:     public MAVLinkPacket pack(){
42:         MAVLinkPacket packet = new MAVLinkPacket();
43:         packet.len = MAVLINK_MSG_LENGTH;
44:         packet.sysid = 255;
45:         packet.compuid = 190;
46:         packet.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;
47:         packet.payload.putByte(target_system);
48:         packet.payload.putByte(control_request);
49:         packet.payload.putByte(version);
50:         for (int i = 0; i < passkey.length; i++) {
51:             packet.payload.putByte(passkey[i]);
52:         }
53:         return packet;
54:     }
55:
56:     /**
57:      * Decode a change_operator_control message into this class fields
58:      * @param payload The message to decode
59:      */
60:     public void unpack(MAVLinkPayload payload) {
61:         payload.resetIndex();
62:         target_system = payload.getByte();
63:         control_request = payload.getByte();

```

```

64:         version = payload.getByte();
65:         for (int i = 0; i < passkey.length; i++) {
66:             passkey[i] = payload.getByte();
67:         }
68:     }
69:
70:     /**
71:      * Constructor for a new message, just initializes the msgid
72:      */
73:     public msg_change_operator_control(){
74:         msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;
75:     }
76:
77:     /**
78:      * Constructor for a new message, initializes the message with the payload
79:      * from a mavlink packet
80:      */
81:
82:     public msg_change_operator_control(MAVLinkPacket mavLinkPacket){
83:         this.sysid = mavLinkPacket.sysid;
84:         this.compuid = mavLinkPacket.compuid;
85:         this.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL;
86:         unpack(mavLinkPacket.payload);
87:         //Log.d("MAVLink", "CHANGE_OPERATOR_CONTROL");
88:         //Log.d("MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL", toString());
89:     }
90:
91:     /**
92:      * Sets the buffer of this message with a string, adds the necessary padding
93:      */
94:     public void setPasskey(String str) {
95:         int len = Math.min(str.length(), 25);
96:         for (int i=0; i<len; i++) {
97:             passkey[i] = (byte) str.charAt(i);
98:         }
99:         for (int i=len; i<25; i++) {
100:             // padding for the rest of
101:             passkey[i] = 0;
102:         }
103:     }
104:
105:     /**
106:      * Gets the message, formatted as a string
107:      */
108:     public String getPasskey() {
109:         String result = "";
110:         for (int i = 0; i < 25; i++) {
111:             if (passkey[i] != 0)
112:                 result = result + (char) passkey[i];
113:             else
114:                 break;
115:         }
116:         return result;
117:     }
118:
119:     /**
120:      * Returns a string with the MSG name and data
121:      */
122:     public String toString(){
123:         return "MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL -"+" target_system:"+target_system+" control_request:"+control_request+" version:"+version+" passkey:"+passkey+"";
124:     }

```



```

1: // MESSAGE CHANGE_OPERATOR_CONTROL_ACK PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Accept / deny control of this MAV
11:  */
12: public class msg_change_operator_control_ack extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK = 6;
15:     public static final int MAVLINK_MSG_LENGTH = 3;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;
17:
18:
19:     /**
20:      * ID of the GCS this message
21:      */
22:     public byte gcs_system_id;
23:     /**
24:      * 0: request control of this MAV, 1: Release control of this MAV
25:      */
26:     public byte control_request;
27:     /**
28:      * 0: ACK, 1: NACK: Wrong passkey, 2: NACK: Unsupported passkey encryption
29:      * method, 3: NACK: Already under control
30:      */
31:     public byte ack;
32:
33:     /**
34:      * Generates the payload for a mavlink message for a message of this type
35:      * @return
36:      */
37:     public MAVLinkPacket pack(){
38:         MAVLinkPacket packet = new MAVLinkPacket();
39:         packet.len = MAVLINK_MSG_LENGTH;
40:         packet.sysid = 255;
41:         packet.compuid = 190;
42:         packet.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;
43:         packet.payload.putByte(gcs_system_id);
44:         packet.payload.putByte(control_request);
45:         packet.payload.putByte(ack);
46:         return packet;
47:     }
48:
49:     /**
50:      * Decode a change_operator_control_ack message into this class fields
51:      *
52:      * @param payload The message to decode
53:      */
54:     public void unpack(MAVLinkPayload payload) {
55:         payload.resetIndex();
56:         gcs_system_id = payload.getByte();
57:         control_request = payload.getByte();
58:         ack = payload.getByte();
59:     }
60:
61:     /**
62:      * Constructor for a new message, just initializes the msgid
63:      */
64:     public msg_change_operator_control_ack(){
65:         msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;
66:     }
67:
68:     /**
69:      * Constructor for a new message, initializes the message with the payload
70:      * from a mavlink packet
71:      */
72:     public msg_change_operator_control_ack(MAVLinkPacket mavLinkPacket){
73:         this.sysid = mavLinkPacket.sysid;
74:         this.compuid = mavLinkPacket.compuid;
75:         this.msgid = MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK;
76:         unpack(mavLinkPacket.payload);
77:         //Log.d("MAVLink", "CHANGE_OPERATOR_CONTROL_ACK");
78:         //Log.d("MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK", toString());
79:     }
80:
81:
82:     /**
83:      * Returns a string with the MSG name and data
84:      */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_CHANGE_OPERATOR_CONTROL_ACK -"+" gcs_system_id:"+gcs_system_id+" control_request:"+control_request+" ack:"+ack+"";
87:     }
88: }

```



```

1: // MESSAGE COMMAND_ACK PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Report status of a command. Includes feedback whether the command was executed.
11:  */
12: public class msg_command_ack extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_COMMAND_ACK = 77;
15:     public static final int MAVLINK_MSG_LENGTH = 3;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_COMMAND_ACK;
17:
18:
19:     /**
20:      * Command ID, as defined by MAV_CMD enum.
21:      */
22:     public short command;
23:     /**
24:      * See MAV_RESULT enum
25:      */
26:     public byte result;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compid = 190;
37:         packet.msgid = MAVLINK_MSG_ID_COMMAND_ACK;
38:         packet.payload.putShort(command);
39:         packet.payload.putByte(result);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a command_ack message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         command = payload.getShort();
51:         result = payload.getByte();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_command_ack(){
58:         msgid = MAVLINK_MSG_ID_COMMAND_ACK;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      *
65:      */
66:     public msg_command_ack(MAVLinkPacket mavLinkPacket){
67:         this.sysid = mavLinkPacket.sysid;

```

```

68:         this.compid = mavLinkPacket.compid;
69:         this.msgid = MAVLINK_MSG_ID_COMMAND_ACK;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "COMMAND_ACK");
72:         //Log.d("MAVLINK_MSG_ID_COMMAND_ACK", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_COMMAND_ACK -" + " command:" + command + " result:" + result +"";
81:     }
82: }

```



```

1: // MESSAGE COMMAND_LONG PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Send a command with up to four parameters to the MAV
11:  */
12: public class msg_command_long extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_COMMAND_LONG = 76;
15:     public static final int MAVLINK_MSG_LENGTH = 33;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_COMMAND_LONG;
17:
18:
19:     /**
20:      * Parameter 1, as defined by MAV_CMD enum.
21:      */
22:     public float param1;
23:     /**
24:      * Parameter 2, as defined by MAV_CMD enum.
25:      */
26:     public float param2;
27:     /**
28:      * Parameter 3, as defined by MAV_CMD enum.
29:      */
30:     public float param3;
31:     /**
32:      * Parameter 4, as defined by MAV_CMD enum.
33:      */
34:     public float param4;
35:     /**
36:      * Parameter 5, as defined by MAV_CMD enum.
37:      */
38:     public float param5;
39:     /**
40:      * Parameter 6, as defined by MAV_CMD enum.
41:      */
42:     public float param6;
43:     /**
44:      * Parameter 7, as defined by MAV_CMD enum.
45:      */
46:     public float param7;
47:     /**
48:      * Command ID, as defined by MAV_CMD enum.
49:      */
50:     public short command;
51:     /**
52:      * System which should execute the command
53:      */
54:     public byte target_system;
55:     /**
56:      * Component which should execute the command, 0 for all components
57:      */
58:     public byte target_component;
59:     /**
60:      * 0: First transmission of this command. 1-255: Confirmation transmissions
61:      * (e.g. for kill command)
62:      */
63:     public byte confirmation;
64:
65:     /**
66:      * Generates the payload for a mavlink message for a message of this type
67:      * @return

```

```

67:     */
68:     public MAVLinkPacket pack(){
69:         MAVLinkPacket packet = new MAVLinkPacket();
70:         packet.len = MAVLINK_MSG_LENGTH;
71:         packet.sysid = 255;
72:         packet.compuid = 190;
73:         packet.msgid = MAVLINK_MSG_ID_COMMAND_LONG;
74:         packet.payload.putFloat(param1);
75:         packet.payload.putFloat(param2);
76:         packet.payload.putFloat(param3);
77:         packet.payload.putFloat(param4);
78:         packet.payload.putFloat(param5);
79:         packet.payload.putFloat(param6);
80:         packet.payload.putFloat(param7);
81:         packet.payload.putShort(command);
82:         packet.payload.putByte(target_system);
83:         packet.payload.putByte(target_component);
84:         packet.payload.putByte(confirmation);
85:         return packet;
86:     }
87:
88:     /**
89:      * Decode a command_long message into this class fields
90:      */
91:     * @param payload The message to decode
92:     */
93:     public void unpack(MAVLinkPayload payload) {
94:         payload.resetIndex();
95:         param1 = payload.getFloat();
96:         param2 = payload.getFloat();
97:         param3 = payload.getFloat();
98:         param4 = payload.getFloat();
99:         param5 = payload.getFloat();
100:        param6 = payload.getFloat();
101:        param7 = payload.getFloat();
102:        command = payload.getShort();
103:        target_system = payload.getByte();
104:        target_component = payload.getByte();
105:        confirmation = payload.getByte();
106:    }
107:
108:    /**
109:     * Constructor for a new message, just initializes the msgid
110:     */
111:    public msg_command_long(){
112:        msgid = MAVLINK_MSG_ID_COMMAND_LONG;
113:    }
114:
115:    /**
116:     * Constructor for a new message, initializes the message with the payload
117:     * from a mavlink packet
118:     */
119:    /**
120:     * public msg_command_long(MAVLinkPacket mavLinkPacket){
121:         this.sysid = mavLinkPacket.sysid;
122:         this.compuid = mavLinkPacket.compuid;
123:         this.msgid = MAVLINK_MSG_ID_COMMAND_LONG;
124:         unpack(mavLinkPacket.payload);
125:         //Log.d("MAVLink", "COMMAND_LONG");
126:         //Log.d("MAVLINK_MSG_ID_COMMAND_LONG", toString());
127:     }
128:
129:
130:    /**
131:     * Returns a string with the MSG name and data
132:     */
133:    public String toString(){

```

```
134:         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+"";
135:     }
136: }
```



```

1: // MESSAGE DATA16 PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Data packet, size 16
11:  */
12: public class msg_data16 extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_DATA16 = 169;
15:     public static final int MAVLINK_MSG_LENGTH = 18;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_DATA16;
17:
18:
19:     /**
20:      * data type
21:      */
22:     public byte type;
23:     /**
24:      * data length
25:      */
26:     public byte len;
27:     /**
28:      * raw data
29:      */
30:     public byte data[] = new byte[16];
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_DATA16;
42:         packet.payload.putByte(type);
43:         packet.payload.putByte(len);
44:         for (int i = 0; i < data.length; i++) {
45:             packet.payload.putByte(data[i]);
46:         }
47:         return packet;
48:     }
49:
50:     /**
51:      * Decode a data16 message into this class fields
52:      *
53:      * @param payload The message to decode
54:      */
55:     public void unpack(MAVLinkPayload payload) {
56:         payload.resetIndex();
57:         type = payload.getByte();
58:         len = payload.getByte();
59:         for (int i = 0; i < data.length; i++) {
60:             data[i] = payload.getByte();
61:         }
62:     }
63:
64:     /**
65:      * Constructor for a new message, just initializes the msgid
66:      */
67:     public msg_data16(){

```

```

68:         msgid = MAVLINK_MSG_ID_DATA16;
69:     }
70:
71:     /**
72:      * Constructor for a new message, initializes the message with the payload
73:      * from a mavlink packet
74:      */
75:     public msg_data16(MAVLinkPacket mavLinkPacket){
76:         this.sysid = mavLinkPacket.sysid;
77:         this.compid = mavLinkPacket.compid;
78:         this.msgid = MAVLINK_MSG_ID_DATA16;
79:         unpack(mavLinkPacket.payload);
80:         //Log.d("MAVLink", "DATA16");
81:         //Log.d("MAVLINK_MSG_ID_DATA16", toString());
82:     }
83:
84:
85:
86:     /**
87:      * Returns a string with the MSG name and data
88:      */
89:     public String toString(){
90:         return "MAVLINK_MSG_ID_DATA16 -"+" type:"+type+" len:"+len+" data:"+data+"
";
91:     }
92: }

```



```

1:  // MESSAGE DATA32 PACKING
2:  package com.MAVLink.Messages.ardupilotmega;
3:
4:  import com.MAVLink.Messages.MAVLinkMessage;
5:  import com.MAVLink.Messages.MAVLinkPayload;
6:  import com.MAVLink.Messages.MAVLinkPacket;
7:  //import android.util.Log;
8:
9:  /**
10:   * Data packet, size 32
11:   */
12:  public class msg_data32 extends MAVLinkMessage{
13:
14:      public static final int MAVLINK_MSG_ID_DATA32 = 170;
15:      public static final int MAVLINK_MSG_LENGTH = 34;
16:      private static final long serialVersionUID = MAVLINK_MSG_ID_DATA32;
17:
18:
19:      /**
20:       * data type
21:       */
22:      public byte type;
23:      /**
24:       * data length
25:       */
26:      public byte len;
27:      /**
28:       * raw data
29:       */
30:      public byte data[] = new byte[32];
31:
32:      /**
33:       * Generates the payload for a mavlink message for a message of this type
34:       * @return
35:       */
36:      public MAVLinkPacket pack(){
37:          MAVLinkPacket packet = new MAVLinkPacket();
38:          packet.len = MAVLINK_MSG_LENGTH;
39:          packet.sysid = 255;
40:          packet.compid = 190;
41:          packet.msgid = MAVLINK_MSG_ID_DATA32;
42:          packet.payload.putByte(type);
43:          packet.payload.putByte(len);
44:          for (int i = 0; i < data.length; i++) {
45:              packet.payload.putByte(data[i]);
46:          }
47:          return packet;
48:      }
49:
50:      /**
51:       * Decode a data32 message into this class fields
52:       *
53:       * @param payload The message to decode
54:       */
55:      public void unpack(MAVLinkPayload payload) {
56:          payload.resetIndex();
57:          type = payload.getByte();
58:          len = payload.getByte();
59:          for (int i = 0; i < data.length; i++) {
60:              data[i] = payload.getByte();
61:          }
62:      }
63:
64:      /**
65:       * Constructor for a new message, just initializes the msgid
66:       */
67:      public msg_data32(){

```

```

68:          msgid = MAVLINK_MSG_ID_DATA32;
69:      }
70:
71:      /**
72:       * Constructor for a new message, initializes the message with the payload
73:       * from a mavlink packet
74:       */
75:      /**
76:       * public msg_data32(MAVLinkPacket mavLinkPacket){
77:          this.sysid = mavLinkPacket.sysid;
78:          this.compid = mavLinkPacket.compid;
79:          this.msgid = MAVLINK_MSG_ID_DATA32;
80:          unpack(mavLinkPacket.payload);
81:          //Log.d("MAVLink", "DATA32");
82:          //Log.d("MAVLINK_MSG_ID_DATA32", toString());
83:      }
84:
85:
86:      /**
87:       * Returns a string with the MSG name and data
88:       */
89:      public String toString(){
90:          return "MAVLINK_MSG_ID_DATA32 -"+" type:"+type+" len:"+len+" data:"+data+"
";
91:      }
92: }

```



```

1:  // MESSAGE DATA64 PACKING
2:  package com.MAVLink.Messages.ardupilotmega;
3:
4:  import com.MAVLink.Messages.MAVLinkMessage;
5:  import com.MAVLink.Messages.MAVLinkPayload;
6:  import com.MAVLink.Messages.MAVLinkPacket;
7:  //import android.util.Log;
8:
9:  /**
10:   * Data packet, size 64
11:   */
12:  public class msg_data64 extends MAVLinkMessage{
13:
14:      public static final int MAVLINK_MSG_ID_DATA64 = 171;
15:      public static final int MAVLINK_MSG_LENGTH = 66;
16:      private static final long serialVersionUID = MAVLINK_MSG_ID_DATA64;
17:
18:
19:      /**
20:       * data type
21:       */
22:      public byte type;
23:      /**
24:       * data length
25:       */
26:      public byte len;
27:      /**
28:       * raw data
29:       */
30:      public byte data[] = new byte[64];
31:
32:      /**
33:       * Generates the payload for a mavlink message for a message of this type
34:       * @return
35:       */
36:      public MAVLinkPacket pack(){
37:          MAVLinkPacket packet = new MAVLinkPacket();
38:          packet.len = MAVLINK_MSG_LENGTH;
39:          packet.sysid = 255;
40:          packet.compid = 190;
41:          packet.msgid = MAVLINK_MSG_ID_DATA64;
42:          packet.payload.putByte(type);
43:          packet.payload.putByte(len);
44:          for (int i = 0; i < data.length; i++) {
45:              packet.payload.putByte(data[i]);
46:          }
47:          return packet;
48:      }
49:
50:      /**
51:       * Decode a data64 message into this class fields
52:       *
53:       * @param payload The message to decode
54:       */
55:      public void unpack(MAVLinkPayload payload) {
56:          payload.resetIndex();
57:          type = payload.getByte();
58:          len = payload.getByte();
59:          for (int i = 0; i < data.length; i++) {
60:              data[i] = payload.getByte();
61:          }
62:      }
63:
64:      /**
65:       * Constructor for a new message, just initializes the msgid
66:       */
67:      public msg_data64(){

```

```

68:          msgid = MAVLINK_MSG_ID_DATA64;
69:      }
70:
71:      /**
72:       * Constructor for a new message, initializes the message with the payload
73:       * from a mavlink packet
74:       */
75:
76:      public msg_data64(MAVLinkPacket mavLinkPacket){
77:          this.sysid = mavLinkPacket.sysid;
78:          this.compid = mavLinkPacket.compid;
79:          this.msgid = MAVLINK_MSG_ID_DATA64;
80:          unpack(mavLinkPacket.payload);
81:          //Log.d("MAVLink", "DATA64");
82:          //Log.d("MAVLINK_MSG_ID_DATA64", toString());
83:      }
84:
85:
86:      /**
87:       * Returns a string with the MSG name and data
88:       */
89:      public String toString(){
90:          return "MAVLINK_MSG_ID_DATA64 -"+" type:"+type+" len:"+len+" data:"+data+"
";
91:      }
92: }

```



```

1:  // MESSAGE DATA96 PACKING
2:  package com.MAVLink.Messages.ardupilotmega;
3:
4:  import com.MAVLink.Messages.MAVLinkMessage;
5:  import com.MAVLink.Messages.MAVLinkPayload;
6:  import com.MAVLink.Messages.MAVLinkPacket;
7:  //import android.util.Log;
8:
9:  /**
10:   * Data packet, size 96
11:   */
12:  public class msg_data96 extends MAVLinkMessage{
13:
14:      public static final int MAVLINK_MSG_ID_DATA96 = 172;
15:      public static final int MAVLINK_MSG_LENGTH = 98;
16:      private static final long serialVersionUID = MAVLINK_MSG_ID_DATA96;
17:
18:
19:      /**
20:       * data type
21:       */
22:      public byte type;
23:      /**
24:       * data length
25:       */
26:      public byte len;
27:      /**
28:       * raw data
29:       */
30:      public byte data[] = new byte[96];
31:
32:      /**
33:       * Generates the payload for a mavlink message for a message of this type
34:       * @return
35:       */
36:      public MAVLinkPacket pack(){
37:          MAVLinkPacket packet = new MAVLinkPacket();
38:          packet.len = MAVLINK_MSG_LENGTH;
39:          packet.sysid = 255;
40:          packet.compid = 190;
41:          packet.msgid = MAVLINK_MSG_ID_DATA96;
42:          packet.payload.putByte(type);
43:          packet.payload.putByte(len);
44:          for (int i = 0; i < data.length; i++) {
45:              packet.payload.putByte(data[i]);
46:          }
47:          return packet;
48:      }
49:
50:      /**
51:       * Decode a data96 message into this class fields
52:       *
53:       * @param payload The message to decode
54:       */
55:      public void unpack(MAVLinkPayload payload) {
56:          payload.resetIndex();
57:          type = payload.getByte();
58:          len = payload.getByte();
59:          for (int i = 0; i < data.length; i++) {
60:              data[i] = payload.getByte();
61:          }
62:      }
63:
64:      /**
65:       * Constructor for a new message, just initializes the msgid
66:       */
67:      public msg_data96(){

```

```

68:          msgid = MAVLINK_MSG_ID_DATA96;
69:      }
70:
71:      /**
72:       * Constructor for a new message, initializes the message with the payload
73:       * from a mavlink packet
74:       */
75:      public msg_data96(MAVLinkPacket mavLinkPacket){
76:          this.sysid = mavLinkPacket.sysid;
77:          this.compid = mavLinkPacket.compid;
78:          this.msgid = MAVLINK_MSG_ID_DATA96;
79:          unpack(mavLinkPacket.payload);
80:          //Log.d("MAVLink", "DATA96");
81:          //Log.d("MAVLINK_MSG_ID_DATA96", toString());
82:      }
83:
84:
85:
86:      /**
87:       * Returns a string with the MSG name and data
88:       */
89:      public String toString(){
90:          return "MAVLINK_MSG_ID_DATA96 -"+" type:"+type+" len:"+len+" data:"+data+"
";
91:      }
92: }

```



```

1: // MESSAGE DATA_STREAM PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: *
11: */
12: public class msg_data_stream extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_DATA_STREAM = 67;
15:     public static final int MAVLINK_MSG_LENGTH = 4;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_DATA_STREAM;
17:
18:
19:     /**
20:     * The requested interval between two messages of this type
21:     */
22:     public short message_rate;
23:     /**
24:     * The ID of the requested data stream
25:     */
26:     public byte stream_id;
27:     /**
28:     * 1 stream is enabled, 0 stream is stopped.
29:     */
30:     public byte on_off;
31:
32:     /**
33:     * Generates the payload for a mavlink message for a message of this type
34:     * @return
35:     */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_DATA_STREAM;
42:         packet.payload.putShort(message_rate);
43:         packet.payload.putByte(stream_id);
44:         packet.payload.putByte(on_off);
45:         return packet;
46:     }
47:
48:     /**
49:     * Decode a data_stream message into this class fields
50:     *
51:     * @param payload The message to decode
52:     */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         message_rate = payload.getShort();
56:         stream_id = payload.getByte();
57:         on_off = payload.getByte();
58:     }
59:
60:     /**
61:     * Constructor for a new message, just initializes the msgid
62:     */
63:     public msg_data_stream(){
64:         msgid = MAVLINK_MSG_ID_DATA_STREAM;
65:     }
66:
67:     /**

```

```

68:     * Constructor for a new message, initializes the message with the payload
69:     * from a mavlink packet
70:     *
71:     */
72:     public msg_data_stream(MAVLinkPacket mavLinkPacket){
73:         this.sysid = mavLinkPacket.sysid;
74:         this.compid = mavLinkPacket.compid;
75:         this.msgid = MAVLINK_MSG_ID_DATA_STREAM;
76:         unpack(mavLinkPacket.payload);
77:         //Log.d("MAVLink", "DATA_STREAM");
78:         //Log.d("MAVLINK_MSG_ID_DATA_STREAM", toString());
79:     }
80:
81:
82:     /**
83:     * Returns a string with the MSG name and data
84:     */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_DATA_STREAM -"+" message_rate:"+message_rate+" stream_id:"+stream_id+" on_off:"+on_off+"";
87:     }
88: }

```



```

1: // MESSAGE DEBUG_VECT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: *
11: */
12: public class msg_debug_vect extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_DEBUG_VECT = 250;
15:     public static final int MAVLINK_MSG_LENGTH = 30;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_DEBUG_VECT;
17:
18:
19:     /**
20:     * Timestamp
21:     */
22:     public long time_usec;
23:     /**
24:     * x
25:     */
26:     public float x;
27:     /**
28:     * y
29:     */
30:     public float y;
31:     /**
32:     * z
33:     */
34:     public float z;
35:     /**
36:     * Name
37:     */
38:     public byte name[] = new byte[10];
39:
40:     /**
41:     * Generates the payload for a mavlink message for a message of this type
42:     * @return
43:     */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_DEBUG_VECT;
50:         packet.payload.putLong(time_usec);
51:         packet.payload.putFloat(x);
52:         packet.payload.putFloat(y);
53:         packet.payload.putFloat(z);
54:         for (int i = 0; i < name.length; i++) {
55:             packet.payload.putByte(name[i]);
56:         }
57:         return packet;
58:     }
59:
60:     /**
61:     * Decode a debug_vect message into this class fields
62:     *
63:     * @param payload The message to decode
64:     */
65:     public void unpack(MAVLinkPayload payload) {
66:         payload.resetIndex();
67:         time_usec = payload.getLong();

```

```

68:         x = payload.getFloat();
69:         y = payload.getFloat();
70:         z = payload.getFloat();
71:         for (int i = 0; i < name.length; i++) {
72:             name[i] = payload.getByte();
73:         }
74:     }
75:
76:     /**
77:     * Constructor for a new message, just initializes the msgid
78:     */
79:     public msg_debug_vect(){
80:         msgid = MAVLINK_MSG_ID_DEBUG_VECT;
81:     }
82:
83:     /**
84:     * Constructor for a new message, initializes the message with the payload
85:     * from a mavlink packet
86:     */
87:     public msg_debug_vect(MAVLinkPacket mavLinkPacket){
88:         this.sysid = mavLinkPacket.sysid;
89:         this.compid = mavLinkPacket.compid;
90:         this.msgid = MAVLINK_MSG_ID_DEBUG_VECT;
91:         unpack(mavLinkPacket.payload);
92:         //Log.d("MAVLink", "DEBUG_VECT");
93:         //Log.d("MAVLINK_MSG_ID_DEBUG_VECT", toString());
94:     }
95:
96:
97:     /**
98:     * Sets the buffer of this message with a string, adds the necessary padding
99:     */
100:     public void setName(String str) {
101:         int len = Math.min(str.length(), 10);
102:         for (int i=0; i<len; i++) {
103:             name[i] = (byte) str.charAt(i);
104:         }
105:         for (int i=len; i<10; i++) { // padding for the rest of
the buffer
106:             name[i] = 0;
107:         }
108:     }
109:
110:     /**
111:     * Gets the message, formatted as a string
112:     */
113:     public String getName() {
114:         String result = "";
115:         for (int i = 0; i < 10; i++) {
116:             if (name[i] != 0)
117:                 result = result + (char) name[i];
118:             else
119:                 break;
120:         }
121:         return result;
122:     }
123:
124:     /**
125:     * Returns a string with the MSG name and data
126:     */
127:     public String toString(){
128:         return "MAVLINK_MSG_ID_DEBUG_VECT -"+" time_usec:"+time_usec+" x:"+x+" y:"
+y+" z:"+z+" name:"+name+"";
129:     }
130: }

```



```

1: // MESSAGE DIGICAM_CONFIGURE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Configure on-board Camera Control System.
11:  */
12: public class msg_digicam_configure extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_DIGICAM_CONFIGURE = 154;
15:     public static final int MAVLINK_MSG_LENGTH = 15;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_DIGICAM_CONFIG
URE;
17:
18:
19:     /**
20:      * Correspondent value to given extra_param
21:      */
22:     public float extra_value;
23:     /**
24:      * Divisor number //e.g. 1000 means 1/1000 (0 means ignore)
25:      */
26:     public short shutter_speed;
27:     /**
28:      * System ID
29:      */
30:     public byte target_system;
31:     /**
32:      * Component ID
33:      */
34:     public byte target_component;
35:     /**
36:      * Mode enumeration from 1 to N //P, TV, AV, M, Etc (0 means ignore)
37:      */
38:     public byte mode;
39:     /**
40:      * F stop number x 10 //e.g. 28 means 2.8 (0 means ignore)
41:      */
42:     public byte aperture;
43:     /**
44:      * ISO enumeration from 1 to N //e.g. 80, 100, 200, Etc (0 means ignore)
45:      */
46:     public byte iso;
47:     /**
48:      * Exposure type enumeration from 1 to N (0 means ignore)
49:      */
50:     public byte exposure_type;
51:     /**
52:      * Command Identity (incremental loop: 0 to 255)//A command sent multiple t
imes will be executed or pooled just once
53:      */
54:     public byte command_id;
55:     /**
56:      * Main engine cut-off time before camera trigger in seconds/10 (0 means no
cut-off)
57:      */
58:     public byte engine_cut_off;
59:     /**
60:      * Extra parameters enumeration (0 means ignore)
61:      */
62:     public byte extra_param;
63:
64:     /**

```

```

65:     * Generates the payload for a mavlink message for a message of this type
66:     * @return
67:     */
68:     public MAVLinkPacket pack(){
69:         MAVLinkPacket packet = new MAVLinkPacket();
70:         packet.len = MAVLINK_MSG_LENGTH;
71:         packet.sysid = 255;
72:         packet.compid = 190;
73:         packet.msgid = MAVLINK_MSG_ID_DIGICAM_CONFIGURE;
74:         packet.payload.putFloat(extra_value);
75:         packet.payload.putShort(shutter_speed);
76:         packet.payload.putByte(target_system);
77:         packet.payload.putByte(target_component);
78:         packet.payload.putByte(mode);
79:         packet.payload.putByte(aperture);
80:         packet.payload.putByte(iso);
81:         packet.payload.putByte(exposure_type);
82:         packet.payload.putByte(command_id);
83:         packet.payload.putByte(engine_cut_off);
84:         packet.payload.putByte(extra_param);
85:         return packet;
86:     }
87:
88:     /**
89:      * Decode a digicam_configure message into this class fields
90:      */
91:     * @param payload The message to decode
92:     */
93:     public void unpack(MAVLinkPayload payload) {
94:         payload.resetIndex();
95:         extra_value = payload.getFloat();
96:         shutter_speed = payload.getShort();
97:         target_system = payload.getByte();
98:         target_component = payload.getByte();
99:         mode = payload.getByte();
100:         aperture = payload.getByte();
101:         iso = payload.getByte();
102:         exposure_type = payload.getByte();
103:         command_id = payload.getByte();
104:         engine_cut_off = payload.getByte();
105:         extra_param = payload.getByte();
106:     }
107:
108:     /**
109:      * Constructor for a new message, just initializes the msgid
110:      */
111:     public msg_digicam_configure(){
112:         msgid = MAVLINK_MSG_ID_DIGICAM_CONFIGURE;
113:     }
114:
115:     /**
116:      * Constructor for a new message, initializes the message with the payload
117:      * from a mavlink packet
118:      */
119:     */
120:     public msg_digicam_configure(MAVLinkPacket mavLinkPacket){
121:         this.sysid = mavLinkPacket.sysid;
122:         this.compid = mavLinkPacket.compid;
123:         this.msgid = MAVLINK_MSG_ID_DIGICAM_CONFIGURE;
124:         unpack(mavLinkPacket.payload);
125:         //Log.d("MAVLink", "DIGICAM_CONFIGURE");
126:         //Log.d("MAVLINK_MSG_ID_DIGICAM_CONFIGURE", toString());
127:     }
128:
129:
130:     /**
131:      * Returns a string with the MSG name and data

```

```
132:     */
133:     public String toString(){
134:         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_typ
e+" command_id:"+command_id+" engine_cut_off:"+engine_cut_off+" extra_param:"+extra_param
+"";
135:     }
136: }
```

```

1: // MESSAGE DIGICAM_CONTROL PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Control on-board Camera Control System to take shots.
11:  */
12: public class msg_digicam_control extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_DIGICAM_CONTROL = 155;
15:     public static final int MAVLINK_MSG_LENGTH = 13;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_DIGICAM_CONTRO
L;
17:
18:
19:     /**
20:      * Correspondent value to given extra_param
21:      */
22:     public float extra_value;
23:     /**
24:      * System ID
25:      */
26:     public byte target_system;
27:     /**
28:      * Component ID
29:      */
30:     public byte target_component;
31:     /**
32:      * 0: stop, 1: start or keep it up //Session control e.g. show/hide lens
33:      */
34:     public byte session;
35:     /**
36:      * 1 to N //Zoom's absolute position (0 means ignore)
37:      */
38:     public byte zoom_pos;
39:     /**
40:      * -100 to 100 //Zooming step value to offset zoom from the current positio
n
41:      */
42:     public byte zoom_step;
43:     /**
44:      * 0: unlock focus or keep unlocked, 1: lock focus or keep locked, 3: re-lo
ck focus
45:      */
46:     public byte focus_lock;
47:     /**
48:      * 0: ignore, 1: shot or start filming
49:      */
50:     public byte shot;
51:     /**
52:      * Command Identity (incremental loop: 0 to 255)//A command sent multiple t
imes will be executed or pooled just once
53:      */
54:     public byte command_id;
55:     /**
56:      * Extra parameters enumeration (0 means ignore)
57:      */
58:     public byte extra_param;
59:
60:     /**
61:      * Generates the payload for a mavlink message for a message of this type
62:      * @return
63:      */

```

```

64:     public MAVLinkPacket pack(){
65:         MAVLinkPacket packet = new MAVLinkPacket();
66:         packet.len = MAVLINK_MSG_LENGTH;
67:         packet.sysid = 255;
68:         packet.compid = 190;
69:         packet.msgid = MAVLINK_MSG_ID_DIGICAM_CONTROL;
70:         packet.payload.putFloat(extra_value);
71:         packet.payload.putByte(target_system);
72:         packet.payload.putByte(target_component);
73:         packet.payload.putByte(session);
74:         packet.payload.putByte(zoom_pos);
75:         packet.payload.putByte(zoom_step);
76:         packet.payload.putByte(focus_lock);
77:         packet.payload.putByte(shot);
78:         packet.payload.putByte(command_id);
79:         packet.payload.putByte(extra_param);
80:         return packet;
81:     }
82:
83:     /**
84:      * Decode a digicam_control message into this class fields
85:      */
86:     * @param payload The message to decode
87:     */
88:     public void unpack(MAVLinkPayload payload) {
89:         payload.resetIndex();
90:         extra_value = payload.getFloat();
91:         target_system = payload.getByte();
92:         target_component = payload.getByte();
93:         session = payload.getByte();
94:         zoom_pos = payload.getByte();
95:         zoom_step = payload.getByte();
96:         focus_lock = payload.getByte();
97:         shot = payload.getByte();
98:         command_id = payload.getByte();
99:         extra_param = payload.getByte();
100:     }
101:
102:     /**
103:      * Constructor for a new message, just initializes the msgid
104:      */
105:     public msg_digicam_control(){
106:         msgid = MAVLINK_MSG_ID_DIGICAM_CONTROL;
107:     }
108:
109:     /**
110:      * Constructor for a new message, initializes the message with the payload
111:      * from a mavlink packet
112:      */
113:     /**
114:      * Constructor for a new message, initializes the message with the payload
115:      * from a mavlink packet
116:      */
117:     public msg_digicam_control(MAVLinkPacket mavLinkPacket){
118:         this.sysid = mavLinkPacket.sysid;
119:         this.compid = mavLinkPacket.compid;
120:         this.msgid = MAVLINK_MSG_ID_DIGICAM_CONTROL;
121:         unpack(mavLinkPacket.payload);
122:         //Log.d("MAVLink", "DIGICAM_CONTROL");
123:         //Log.d("MAVLINK_MSG_ID_DIGICAM_CONTROL", toString());
124:     }
125:
126:     /**
127:      * Returns a string with the MSG name and data
128:      */
129:     public String toString(){
130:         return "MAVLINK_MSG_ID_DIGICAM_CONTROL -"+" extra_value:"+extra_value+" ta
rget_system:"+target_system+" target_component:"+target_component+" session:"+session+" z
oom_pos:"+zoom_pos+" zoom_step:"+zoom_step+" focus_lock:"+focus_lock+" shot:"+shot+" comm

```

./com/MAVLink/Messages/ardupilotmega/msg_digicam_control.java

Fri Oct 25 14:10:50 2013

2

```
and_id:"+command_id+" extra_param:"+extra_param+"";
129:     }
130: }
```



```

1: // MESSAGE FENCE_FETCH_POINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Request a current fence point from MAV
11:  */
12: public class msg_fence_fetch_point extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_FENCE_FETCH_POINT = 161;
15:     public static final int MAVLINK_MSG_LENGTH = 3;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_FENCE_FETCH_PO
INT;
17:
18:
19:     /**
20:      * System ID
21:      */
22:     public byte target_system;
23:     /**
24:      * Component ID
25:      */
26:     public byte target_component;
27:     /**
28:      * point index (first point is 1, 0 is for return point)
29:      */
30:     public byte idx;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_FENCE_FETCH_POINT;
42:         packet.payload.putByte(target_system);
43:         packet.payload.putByte(target_component);
44:         packet.payload.putByte(idx);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a fence_fetch_point message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         target_system = payload.getByte();
56:         target_component = payload.getByte();
57:         idx = payload.getByte();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_fence_fetch_point(){
64:         msgid = MAVLINK_MSG_ID_FENCE_FETCH_POINT;
65:     }
66:

```

```

67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     public msg_fence_fetch_point(MAVLinkPacket mavLinkPacket){
72:         this.sysid = mavLinkPacket.sysid;
73:         this.compid = mavLinkPacket.compid;
74:         this.msgid = MAVLINK_MSG_ID_FENCE_FETCH_POINT;
75:         unpack(mavLinkPacket.payload);
76:         //Log.d("MAVLink", "FENCE_FETCH_POINT");
77:         //Log.d("MAVLINK_MSG_ID_FENCE_FETCH_POINT", toString());
78:     }
79:
80:
81:
82:     /**
83:      * Returns a string with the MSG name and data
84:      */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_FENCE_FETCH_POINT -"+" target_system:"+target_syste
m+" target_component:"+target_component+" idx:"+idx+"";
87:     }
88: }

```



```

1: // MESSAGE FENCE_POINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * A fence point. Used to set a point when from
11:  * GCS -> MAV. Also used to return a point from MAV -> GCS
12:  */
13: public class msg_fence_point extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_FENCE_POINT = 160;
16:     public static final int MAVLINK_MSG_LENGTH = 12;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_FENCE_POINT;
18:
19:
20:     /**
21:      * Latitude of point
22:      */
23:     public float lat;
24:     /**
25:      * Longitude of point
26:      */
27:     public float lng;
28:     /**
29:      * System ID
30:      */
31:     public byte target_system;
32:     /**
33:      * Component ID
34:      */
35:     public byte target_component;
36:     /**
37:      * point index (first point is 1, 0 is for return point)
38:      */
39:     public byte idx;
40:     /**
41:      * total number of points (for sanity checking)
42:      */
43:     public byte count;
44:
45:     /**
46:      * Generates the payload for a mavlink message for a message of this type
47:      * @return
48:      */
49:     public MAVLinkPacket pack(){
50:         MAVLinkPacket packet = new MAVLinkPacket();
51:         packet.len = MAVLINK_MSG_LENGTH;
52:         packet.sysid = 255;
53:         packet.compid = 190;
54:         packet.msgid = MAVLINK_MSG_ID_FENCE_POINT;
55:         packet.payload.putFloat(lat);
56:         packet.payload.putFloat(lng);
57:         packet.payload.putByte(target_system);
58:         packet.payload.putByte(target_component);
59:         packet.payload.putByte(idx);
60:         packet.payload.putByte(count);
61:         return packet;
62:     }
63:
64:     /**
65:      * Decode a fence_point message into this class fields
66:      *
67:      * @param payload The message to decode

```

```

68:     */
69:     public void unpack(MAVLinkPayload payload) {
70:         payload.resetIndex();
71:         lat = payload.getFloat();
72:         lng = payload.getFloat();
73:         target_system = payload.getByte();
74:         target_component = payload.getByte();
75:         idx = payload.getByte();
76:         count = payload.getByte();
77:     }
78:
79:     /**
80:      * Constructor for a new message, just initializes the msgid
81:      */
82:     public msg_fence_point(){
83:         msgid = MAVLINK_MSG_ID_FENCE_POINT;
84:     }
85:
86:     /**
87:      * Constructor for a new message, initializes the message with the payload
88:      * from a mavlink packet
89:      */
90:     public msg_fence_point(MAVLinkPacket mavLinkPacket){
91:         this.sysid = mavLinkPacket.sysid;
92:         this.compid = mavLinkPacket.compid;
93:         this.msgid = MAVLINK_MSG_ID_FENCE_POINT;
94:         unpack(mavLinkPacket.payload);
95:         //Log.d("MAVLink", "FENCE_POINT");
96:         //Log.d("MAVLINK_MSG_ID_FENCE_POINT", toString());
97:     }
98:
99:
100:
101:     /**
102:      * Returns a string with the MSG name and data
103:      */
104:     public String toString(){
105:         return "MAVLINK_MSG_ID_FENCE_POINT -"+" lat:"+lat+" lng:"+lng+" target_sys
tem:"+target_system+" target_component:"+target_component+" idx:"+idx+" count:"+count+"";
106:     }
107: }

```



```

1: // MESSAGE FENCE_STATUS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Status of geo-fencing. Sent in extended
11:  * status stream when fencing enabled
12:  */
13: public class msg_fence_status extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_FENCE_STATUS = 162;
16:     public static final int MAVLINK_MSG_LENGTH = 8;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_FENCE_STATUS;
18:
19:
20:     /**
21:     * time of last breach in milliseconds since boot
22:     */
23:     public int breach_time;
24:     /**
25:     * number of fence breaches
26:     */
27:     public short breach_count;
28:     /**
29:     * 0 if currently inside fence, 1 if outside
30:     */
31:     public byte breach_status;
32:     /**
33:     * last breach type (see FENCE_BREACH_* enum)
34:     */
35:     public byte breach_type;
36:
37:     /**
38:     * Generates the payload for a mavlink message for a message of this type
39:     * @return
40:     */
41:     public MAVLinkPacket pack(){
42:         MAVLinkPacket packet = new MAVLinkPacket();
43:         packet.len = MAVLINK_MSG_LENGTH;
44:         packet.sysid = 255;
45:         packet.compuid = 190;
46:         packet.msgid = MAVLINK_MSG_ID_FENCE_STATUS;
47:         packet.payload.putInt(breach_time);
48:         packet.payload.putShort(breach_count);
49:         packet.payload.putByte(breach_status);
50:         packet.payload.putByte(breach_type);
51:         return packet;
52:     }
53:
54:     /**
55:     * Decode a fence_status message into this class fields
56:     *
57:     * @param payload The message to decode
58:     */
59:     public void unpack(MAVLinkPayload payload) {
60:         payload.resetIndex();
61:         breach_time = payload.getInt();
62:         breach_count = payload.getShort();
63:         breach_status = payload.getByte();
64:         breach_type = payload.getByte();
65:     }
66:
67:     /**

```

```

68:     * Constructor for a new message, just initializes the msgid
69:     */
70:     public msg_fence_status(){
71:         msgid = MAVLINK_MSG_ID_FENCE_STATUS;
72:     }
73:
74:     /**
75:     * Constructor for a new message, initializes the message with the payload
76:     * from a mavlink packet
77:     */
78:     public msg_fence_status(MAVLinkPacket mavLinkPacket){
79:         this.sysid = mavLinkPacket.sysid;
80:         this.compuid = mavLinkPacket.compuid;
81:         this.msgid = MAVLINK_MSG_ID_FENCE_STATUS;
82:         unpack(mavLinkPacket.payload);
83:         //Log.d("MAVLink", "FENCE_STATUS");
84:         //Log.d("MAVLINK_MSG_ID_FENCE_STATUS", toString());
85:     }
86:
87:
88:
89:     /**
90:     * Returns a string with the MSG name and data
91:     */
92:     public String toString(){
93:         return "MAVLINK_MSG_ID_FENCE_STATUS -"+" breach_time:"+breach_time+" breach
h_count:"+breach_count+" breach_status:"+breach_status+" breach_type:"+breach_type+"";
94:     }
95: }

```



```

1:  // MESSAGE FILE_TRANSFER_DIR_LIST PACKING
2:  package com.MAVLink.Messages.ardupilotmega;
3:
4:  import com.MAVLink.Messages.MAVLinkMessage;
5:  import com.MAVLink.Messages.MAVLinkPayload;
6:  import com.MAVLink.Messages.MAVLinkPacket;
7:  //import android.util.Log;
8:
9:  /**
10:   * Get directory listing
11:   */
12:  public class msg_file_transfer_dir_list extends MAVLinkMessage{
13:
14:      public static final int MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST = 111;
15:      public static final int MAVLINK_MSG_LENGTH = 249;
16:      private static final long serialVersionUID = MAVLINK_MSG_ID_FILE_TRANSFER_
DIR_LIST;
17:
18:
19:      /**
20:       * Unique transfer ID
21:       */
22:      public long transfer_uid;
23:      /**
24:       * Directory path to list
25:       */
26:      public byte dir_path[] = new byte[240];
27:      /**
28:       * RESERVED
29:       */
30:      public byte flags;
31:
32:      /**
33:       * Generates the payload for a mavlink message for a message of this type
34:       * @return
35:       */
36:      public MAVLinkPacket pack(){
37:          MAVLinkPacket packet = new MAVLinkPacket();
38:          packet.len = MAVLINK_MSG_LENGTH;
39:          packet.sysid = 255;
40:          packet.compuid = 190;
41:          packet.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST;
42:          packet.payload.putLong(transfer_uid);
43:          for (int i = 0; i < dir_path.length; i++) {
44:              packet.payload.putByte(dir_path[i]);
45:          }
46:          packet.payload.putByte(flags);
47:          return packet;
48:      }
49:
50:      /**
51:       * Decode a file_transfer_dir_list message into this class fields
52:       *
53:       * @param payload The message to decode
54:       */
55:      public void unpack(MAVLinkPayload payload) {
56:          payload.resetIndex();
57:          transfer_uid = payload.getLong();
58:          for (int i = 0; i < dir_path.length; i++) {
59:              dir_path[i] = payload.getByte();
60:          }
61:          flags = payload.getByte();
62:      }
63:
64:      /**
65:       * Constructor for a new message, just initializes the msgid
66:       */

```

```

67:  public msg_file_transfer_dir_list(){
68:      msgid = MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST;
69:  }
70:
71:  /**
72:   * Constructor for a new message, initializes the message with the payload
73:   * from a mavlink packet
74:   */
75:  public msg_file_transfer_dir_list(MAVLinkPacket mavLinkPacket){
76:      this.sysid = mavLinkPacket.sysid;
77:      this.compuid = mavLinkPacket.compuid;
78:      this.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST;
79:      unpack(mavLinkPacket.payload);
80:      //Log.d("MAVLink", "FILE_TRANSFER_DIR_LIST");
81:      //Log.d("MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST", toString());
82:  }
83:
84:  /**
85:   * Sets the buffer of this message with a string, adds the necessary padding
86:   */
87:  public void setDir_Path(String str) {
88:      int len = Math.min(str.length(), 240);
89:      for (int i=0; i<len; i++) {
90:          dir_path[i] = (byte) str.charAt(i);
91:      }
92:      for (int i=len; i<240; i++) {
93:          // padding for the rest of
the buffer
94:          dir_path[i] = 0;
95:      }
96:  }
97:
98:  /**
99:   * Gets the message, formatted as a string
100:   */
101:  public String getDir_Path() {
102:      String result = "";
103:      for (int i = 0; i < 240; i++) {
104:          if (dir_path[i] != 0)
105:              result = result + (char) dir_path[i];
106:          else
107:              break;
108:      }
109:      return result;
110:  }
111:
112:  /**
113:   * Returns a string with the MSG name and data
114:   */
115:  public String toString(){
116:      return "MAVLINK_MSG_ID_FILE_TRANSFER_DIR_LIST -" + transfer_uid + "transfer
uid:" + dir_path + " flags:" + flags + " ";
117:  }
118:  }

```


./com/MAVLink/Messages/ardupilotmega/msg_file_transfer_res.java

```
1: // MESSAGE FILE_TRANSFER_RES PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * File transfer result
11:  */
12: public class msg_file_transfer_res extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_FILE_TRANSFER_RES = 112;
15:     public static final int MAVLINK_MSG_LENGTH = 9;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_FILE_TRANSFER_
RES;
17:
18:
19:     /**
20:      * Unique transfer ID
21:      */
22:     public long transfer_uid;
23:     /**
24:      * 0: OK, 1: not permitted, 2: bad path / file name, 3: no space left on de
vice
25:      */
26:     public byte result;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compuid = 190;
37:         packet.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_RES;
38:         packet.payload.putLong(transfer_uid);
39:         packet.payload.putByte(result);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a file_transfer_res message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         transfer_uid = payload.getLong();
51:         result = payload.getByte();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_file_transfer_res(){
58:         msgid = MAVLINK_MSG_ID_FILE_TRANSFER_RES;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      *
65:      */
```

Fri Oct 25 14:10:50 2013

1

```
66:     public msg_file_transfer_res(MAVLinkPacket mavLinkPacket){
67:         this.sysid = mavLinkPacket.sysid;
68:         this.compuid = mavLinkPacket.compuid;
69:         this.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_RES;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "FILE_TRANSFER_RES");
72:         //Log.d("MAVLINK_MSG_ID_FILE_TRANSFER_RES", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_FILE_TRANSFER_RES -"+" transfer_uid:"+transfer_uid+
" result:"+result+"";
81:     }
82: }
```



```

1: // MESSAGE FILE_TRANSFER_START PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Begin file transfer
11:  */
12: public class msg_file_transfer_start extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_FILE_TRANSFER_START = 110;
15:     public static final int MAVLINK_MSG_LENGTH = 254;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_FILE_TRANSFER_
START;
17:
18:
19:     /**
20:      * Unique transfer ID
21:      */
22:     public long transfer_uid;
23:     /**
24:      * File size in bytes
25:      */
26:     public int file_size;
27:     /**
28:      * Destination path
29:      */
30:     public byte dest_path[] = new byte[240];
31:     /**
32:      * Transfer direction: 0: from requester, 1: to requester
33:      */
34:     public byte direction;
35:     /**
36:      * RESERVED
37:      */
38:     public byte flags;
39:
40:     /**
41:      * Generates the payload for a mavlink message for a message of this type
42:      * @return
43:      */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_START;
50:         packet.payload.putLong(transfer_uid);
51:         packet.payload.putInt(file_size);
52:         for (int i = 0; i < dest_path.length; i++) {
53:             packet.payload.putByte(dest_path[i]);
54:         }
55:         packet.payload.putByte(direction);
56:         packet.payload.putByte(flags);
57:         return packet;
58:     }
59:
60:     /**
61:      * Decode a file_transfer_start message into this class fields
62:      *
63:      * @param payload The message to decode
64:      */
65:     public void unpack(MAVLinkPayload payload) {
66:         payload.resetIndex();

```

```

67:         transfer_uid = payload.getLong();
68:         file_size = payload.getInt();
69:         for (int i = 0; i < dest_path.length; i++) {
70:             dest_path[i] = payload.getByte();
71:         }
72:         direction = payload.getByte();
73:         flags = payload.getByte();
74:     }
75:
76:     /**
77:      * Constructor for a new message, just initializes the msgid
78:      */
79:     public msg_file_transfer_start(){
80:         msgid = MAVLINK_MSG_ID_FILE_TRANSFER_START;
81:     }
82:
83:     /**
84:      * Constructor for a new message, initializes the message with the payload
85:      * from a mavlink packet
86:      */
87:
88:     public msg_file_transfer_start(MAVLinkPacket mavLinkPacket){
89:         this.sysid = mavLinkPacket.sysid;
90:         this.compid = mavLinkPacket.compid;
91:         this.msgid = MAVLINK_MSG_ID_FILE_TRANSFER_START;
92:         unpack(mavLinkPacket.payload);
93:         //Log.d("MAVLink", "FILE_TRANSFER_START");
94:         //Log.d("MAVLINK_MSG_ID_FILE_TRANSFER_START", toString());
95:     }
96:
97:     /**
98:      * Sets the buffer of this message with a string, adds the necessary padding
99:      */
100:     public void setDest_Path(String str) {
101:         int len = Math.min(str.length(), 240);
102:         for (int i=0; i<len; i++) {
103:             dest_path[i] = (byte) str.charAt(i);
104:         }
105:         for (int i=len; i<240; i++) {
106:             dest_path[i] = 0;
107:         }
108:     }
109:
110:     /**
111:      * Gets the message, formatted as a string
112:      */
113:     public String getDest_Path() {
114:         String result = "";
115:         for (int i = 0; i < 240; i++) {
116:             if (dest_path[i] != 0)
117:                 result = result + (char) dest_path[i];
118:             else
119:                 break;
120:         }
121:         return result;
122:     }
123:
124:     /**
125:      * Returns a string with the MSG name and data
126:      */
127:     public String toString(){
128:         return "MAVLINK_MSG_ID_FILE_TRANSFER_START -"+" transfer_uid:"+transfer_uid+" file_size:"+file_size+" dest_path:"+dest_path+" direction:"+direction+" flags:"+flags+"";
129:     }
130: }

```



```

1: // MESSAGE GLOBAL_POSITION_INT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * The filtered global position (e.g. fused GPS and accelerometers). The position is
11:  * in GPS-frame (right-handed, Z-up). It is designed as scaled integer message since the resolution of float
12:  * is not sufficient.
13:  */
14: public class msg_global_position_int extends MAVLinkMessage{
15:
16:     public static final int MAVLINK_MSG_ID_GLOBAL_POSITION_INT = 33;
17:     public static final int MAVLINK_MSG_LENGTH = 28;
18:     private static final long serialVersionUID = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;
19:
20:     /**
21:      * Timestamp (milliseconds since system boot)
22:      */
23:     public int time_boot_ms;
24:     /**
25:      * Latitude, expressed as * 1E7
26:      */
27:     public int lat;
28:     /**
29:      * Longitude, expressed as * 1E7
30:      */
31:     public int lon;
32:     /**
33:      * Altitude in meters, expressed as * 1000 (millimeters), above MSL
34:      */
35:     public int alt;
36:     /**
37:      * Altitude above ground in meters, expressed as * 1000 (millimeters)
38:      */
39:     public int relative_alt;
40:     /**
41:      * Ground X Speed (Latitude), expressed as m/s * 100
42:      */
43:     public short vx;
44:     /**
45:      * Ground Y Speed (Longitude), expressed as m/s * 100
46:      */
47:     public short vy;
48:     /**
49:      * Ground Z Speed (Altitude), expressed as m/s * 100
50:      */
51:     public short vz;
52:     /**
53:      * Compass heading in degrees * 100, 0.0..359.99 degrees. If unknown, set to
54:      */
55:     public short hdg;
56:
57:     /**
58:      * Generates the payload for a mavlink message for a message of this type
59:      * @return
60:      */
61:     public MAVLinkPacket pack(){
62:         MAVLinkPacket packet = new MAVLinkPacket();
63:         packet.len = MAVLINK_MSG_LENGTH;

```

```

64:         packet.sysid = 255;
65:         packet.compuid = 190;
66:         packet.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;
67:         packet.payload.putInt(time_boot_ms);
68:         packet.payload.putInt(lat);
69:         packet.payload.putInt(lon);
70:         packet.payload.putInt(alt);
71:         packet.payload.putInt(relative_alt);
72:         packet.payload.putShort(vx);
73:         packet.payload.putShort(vy);
74:         packet.payload.putShort(vz);
75:         packet.payload.putShort(hdg);
76:         return packet;
77:     }
78:
79:     /**
80:      * Decode a global_position_int message into this class fields
81:      */
82:     * @param payload The message to decode
83:     */
84:     public void unpack(MAVLinkPayload payload) {
85:         payload.resetIndex();
86:         time_boot_ms = payload.getInt();
87:         lat = payload.getInt();
88:         lon = payload.getInt();
89:         alt = payload.getInt();
90:         relative_alt = payload.getInt();
91:         vx = payload.getShort();
92:         vy = payload.getShort();
93:         vz = payload.getShort();
94:         hdg = payload.getShort();
95:     }
96:
97:     /**
98:      * Constructor for a new message, just initializes the msgid
99:      */
100:     public msg_global_position_int(){
101:         msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;
102:     }
103:
104:     /**
105:      * Constructor for a new message, initializes the message with the payload
106:      * from a mavlink packet
107:      */
108:
109:     public msg_global_position_int(MAVLinkPacket mavLinkPacket){
110:         this.sysid = mavLinkPacket.sysid;
111:         this.compuid = mavLinkPacket.compuid;
112:         this.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_INT;
113:         unpack(mavLinkPacket.payload);
114:         //Log.d("MAVLink", "GLOBAL_POSITION_INT");
115:         //Log.d("MAVLINK_MSG_ID_GLOBAL_POSITION_INT", toString());
116:     }
117:
118:
119:     /**
120:      * Returns a string with the MSG name and data
121:      */
122:     public String toString(){
123:         return "MAVLINK_MSG_ID_GLOBAL_POSITION_INT -" + time_boot_ms + "time_boot_ms:" + lat + "lon:" + lon + "alt:" + alt + "relative_alt:" + relative_alt + "vx:" + vx + "vy:" + vy + "vz:" + vz + "hdg:" + hdg + "";
124:     }
125: }

```



```

1: // MESSAGE GLOBAL_POSITION_SETPOINT_INT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Transmit the current local setpoint of the controller to other MAVs (collision a
11:  * voidance) and to the GCS.
12:  */
13: public class msg_global_position_setpoint_int extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT = 52;
16:     public static final int MAVLINK_MSG_LENGTH = 15;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_GLOBAL_POSITIO
18: N_SETPOINT_INT;
19:
20:     /**
21:     * WGS84 Latitude position in degrees * 1E7
22:     */
23:     public int latitude;
24:     /**
25:     * WGS84 Longitude position in degrees * 1E7
26:     */
27:     public int longitude;
28:     /**
29:     * WGS84 Altitude in meters * 1000 (positive for up)
30:     */
31:     public int altitude;
32:     /**
33:     * Desired yaw angle in degrees * 100
34:     */
35:     public short yaw;
36:     /**
37:     * Coordinate frame - valid values are only MAV_FRAME_GLOBAL or MAV_FRAME_G
38:     LOCAL_RELATIVE_ALT
39:     */
40:     public byte coordinate_frame;
41:
42:     /**
43:     * Generates the payload for a mavlink message for a message of this type
44:     * @return
45:     */
46:     public MAVLinkPacket pack(){
47:         MAVLinkPacket packet = new MAVLinkPacket();
48:         packet.len = MAVLINK_MSG_LENGTH;
49:         packet.sysid = 255;
50:         packet.compid = 190;
51:         packet.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT;
52:         packet.payload.putInt(latitude);
53:         packet.payload.putInt(longitude);
54:         packet.payload.putInt(altitude);
55:         packet.payload.putShort(yaw);
56:         packet.payload.putByte(coordinate_frame);
57:         return packet;
58:     }
59:
60:     /**
61:     * Decode a global_position_setpoint_int message into this class fields
62:     * @param payload The message to decode
63:     */
64:     public void unpack(MAVLinkPayload payload) {
65:         latitude = payload.getInt();
66:         longitude = payload.getInt();
67:         altitude = payload.getInt();
68:         yaw = payload.getShort();
69:         coordinate_frame = payload.getByte();
70:     }
71:
72:     /**
73:     * Constructor for a new message, just initializes the msgid
74:     */
75:     public msg_global_position_setpoint_int(){
76:         msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT;
77:     }
78:
79:     /**
80:     * Constructor for a new message, initializes the message with the payload
81:     * from a mavlink packet
82:     */
83:     public msg_global_position_setpoint_int(MAVLinkPacket mavLinkPacket){
84:         this.sysid = mavLinkPacket.sysid;
85:         this.compid = mavLinkPacket.compid;
86:         this.msgid = MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT;
87:         unpack(mavLinkPacket.payload);
88:         //Log.d("MAVLink", "GLOBAL_POSITION_SETPOINT_INT");
89:         //Log.d("MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT", toString());
90:     }
91:
92:     /**
93:     * Returns a string with the MSG name and data
94:     */
95:     public String toString(){
96:         return "MAVLINK_MSG_ID_GLOBAL_POSITION_SETPOINT_INT -" + latitude + "latitu
97: de+" longitude+" +longitude+" altitude:" +altitude+" yaw:" +yaw+" coordinate_frame:" +coordin
98: ate_frame+"";
99:     }
100: }

```



```

1: // MESSAGE GLOBAL_VISION_POSITION_ESTIMATE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: *
11: */
12: public class msg_global_vision_position_estimate extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE = 1
01:
15:     public static final int MAVLINK_MSG_LENGTH = 32;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_GLOBAL_VISION_
POSITION_ESTIMATE;
17:
18:
19:     /**
20:     * Timestamp (microseconds, synced to UNIX time or since system boot)
21:     */
22:     public long usec;
23:     /**
24:     * Global X position
25:     */
26:     public float x;
27:     /**
28:     * Global Y position
29:     */
30:     public float y;
31:     /**
32:     * Global Z position
33:     */
34:     public float z;
35:     /**
36:     * Roll angle in rad
37:     */
38:     public float roll;
39:     /**
40:     * Pitch angle in rad
41:     */
42:     public float pitch;
43:     /**
44:     * Yaw angle in rad
45:     */
46:     public float yaw;
47:
48:     /**
49:     * Generates the payload for a mavlink message for a message of this type
50:     * @return
51:     */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE;
58:         packet.payload.putLong(usec);
59:         packet.payload.putFloat(x);
60:         packet.payload.putFloat(y);
61:         packet.payload.putFloat(z);
62:         packet.payload.putFloat(roll);
63:         packet.payload.putFloat(pitch);
64:         packet.payload.putFloat(yaw);
65:         return packet;
66:     }
67:
68:     /**
69:     * Decode a global_vision_position_estimate message into this class fields
70:     *
71:     * @param payload The message to decode
72:     */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         usec = payload.getLong();
76:         x = payload.getFloat();
77:         y = payload.getFloat();
78:         z = payload.getFloat();
79:         roll = payload.getFloat();
80:         pitch = payload.getFloat();
81:         yaw = payload.getFloat();
82:     }
83:
84:     /**
85:     * Constructor for a new message, just initializes the msgid
86:     */
87:     public msg_global_vision_position_estimate(){
88:         msgid = MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE;
89:     }
90:
91:     /**
92:     * Constructor for a new message, initializes the message with the payload
93:     * from a mavlink packet
94:     */
95:
96:     public msg_global_vision_position_estimate(MAVLinkPacket mavLinkPacket){
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compid = mavLinkPacket.compid;
99:         this.msgid = MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "GLOBAL_VISION_POSITION_ESTIMATE");
102:         //Log.d("MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE", toString());
103:     }
104:
105:
106:     /**
107:     * Returns a string with the MSG name and data
108:     */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_GLOBAL_VISION_POSITION_ESTIMATE -"+" usec:"+usec+"
x:"+x+" y:"+y+" z:"+z+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+"";
111:     }
112: }

```



```

1: // MESSAGE GPS_GLOBAL_ORIGIN PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Once the MAV sets a new GPS-Local correspondence, this message announces the ori
gin (0,0,0) position
11:  */
12: public class msg_gps_global_origin extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN = 49;
15:     public static final int MAVLINK_MSG_LENGTH = 12;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_GPS_GLOBAL_ORI
GIN;
17:
18:
19:     /**
20:      * Latitude (WGS84), expressed as * 1E7
21:      */
22:     public int latitude;
23:     /**
24:      * Longitude (WGS84), expressed as * 1E7
25:      */
26:     public int longitude;
27:     /**
28:      * Altitude(WGS84), expressed as * 1000
29:      */
30:     public int altitude;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compuid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN;
42:         packet.payload.putInt(latitude);
43:         packet.payload.putInt(longitude);
44:         packet.payload.putInt(altitude);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a gps_global_origin message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         latitude = payload.getInt();
56:         longitude = payload.getInt();
57:         altitude = payload.getInt();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_gps_global_origin(){
64:         msgid = MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN;
65:     }

```

```

66:
67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     public msg_gps_global_origin(MAVLinkPacket mavLinkPacket){
72:         this.sysid = mavLinkPacket.sysid;
73:         this.compuid = mavLinkPacket.compuid;
74:         this.msgid = MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN;
75:         unpack(mavLinkPacket.payload);
76:         //Log.d("MAVLink", "GPS_GLOBAL_ORIGIN");
77:         //Log.d("MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN", toString());
78:     }
79:
80:
81:     /**
82:      * Returns a string with the MSG name and data
83:      */
84:     public String toString(){
85:         return "MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN -"+" latitude:"+latitude+" longit
ude:"+longitude+" altitude:"+altitude+"";
86:     }
87:
88: }

```



```

1: // MESSAGE GPS_RAW_INT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * The global position, as returned by the Global Positioning System (GPS). This is
11:  * NOT the global position estimate of the sytem, but rather a RAW se
nsor value. See message GLOBAL_POSITION for the global position estimate. Coordinate fram
e is right-handed, Z-axis up (GPS frame).
12:  */
13: public class msg_gps_raw_int extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_GPS_RAW_INT = 24;
16:     public static final int MAVLINK_MSG_LENGTH = 30;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_GPS_RAW_INT;
18:
19:
20:     /**
21:      * Timestamp (microseconds since UNIX epoch or microseconds since system bo
ot)
22:      */
23:     public long time_usec;
24:     /**
25:      * Latitude in 1E7 degrees
26:      */
27:     public int lat;
28:     /**
29:      * Longitude in 1E7 degrees
30:      */
31:     public int lon;
32:     /**
33:      * Altitude in 1E3 meters (millimeters) above MSL
34:      */
35:     public int alt;
36:     /**
37:      * GPS HDOP horizontal dilution of position in cm (m*100). If unknown, set
to: 65535
38:      */
39:     public short eph;
40:     /**
41:      * GPS VDOP horizontal dilution of position in cm (m*100). If unknown, set
to: 65535
42:      */
43:     public short epv;
44:     /**
45:      * GPS ground speed (m/s * 100). If unknown, set to: 65535
46:      */
47:     public short vel;
48:     /**
49:      * Course over ground (NOT heading, but direction of movement) in degrees *
100, 0.0..359.99 degrees. If unknown, set to: 65535
50:      */
51:     public short cog;
52:     /**
53:      * 0-1: no fix, 2: 2D fix, 3: 3D fix. Some applications will not use the va
lue of this field unless it is at least two, so always correctly fill in the fix.
54:      */
55:     public byte fix_type;
56:     /**
57:      * Number of satellites visible. If unknown, set to 255
58:      */
59:     public byte satellites_visible;
60:

```

```

61:     /**
62:      * Generates the payload for a mavlink message for a message of this type
63:      * @return
64:      */
65:     public MAVLinkPacket pack(){
66:         MAVLinkPacket packet = new MAVLinkPacket();
67:         packet.len = MAVLINK_MSG_LENGTH;
68:         packet.sysid = 255;
69:         packet.compid = 190;
70:         packet.msgid = MAVLINK_MSG_ID_GPS_RAW_INT;
71:         packet.payload.putLong(time_usec);
72:         packet.payload.putInt(lat);
73:         packet.payload.putInt(lon);
74:         packet.payload.putInt(alt);
75:         packet.payload.putShort(eph);
76:         packet.payload.putShort(epv);
77:         packet.payload.putShort(vel);
78:         packet.payload.putShort(cog);
79:         packet.payload.putByte(fix_type);
80:         packet.payload.putByte(satellites_visible);
81:         return packet;
82:     }
83:
84:     /**
85:      * Decode a gps_raw_int message into this class fields
86:      *
87:      * @param payload The message to decode
88:      */
89:     public void unpack(MAVLinkPayload payload) {
90:         payload.resetIndex();
91:         time_usec = payload.getLong();
92:         lat = payload.getInt();
93:         lon = payload.getInt();
94:         alt = payload.getInt();
95:         eph = payload.getShort();
96:         epv = payload.getShort();
97:         vel = payload.getShort();
98:         cog = payload.getShort();
99:         fix_type = payload.getByte();
100:         satellites_visible = payload.getByte();
101:     }
102:
103:     /**
104:      * Constructor for a new message, just initializes the msgid
105:      */
106:     public msg_gps_raw_int(){
107:         msgid = MAVLINK_MSG_ID_GPS_RAW_INT;
108:     }
109:
110:     /**
111:      * Constructor for a new message, initializes the message with the payload
112:      * from a mavlink packet
113:      */
114:     public msg_gps_raw_int(MAVLinkPacket mavLinkPacket){
115:         this.sysid = mavLinkPacket.sysid;
116:         this.compid = mavLinkPacket.compid;
117:         this.msgid = MAVLINK_MSG_ID_GPS_RAW_INT;
118:         unpack(mavLinkPacket.payload);
119:         //Log.d("MAVLink", "GPS_RAW_INT");
120:         //Log.d("MAVLINK_MSG_ID_GPS_RAW_INT", toString());
121:     }
122:
123:
124:
125:     /**
126:      * Returns a string with the MSG name and data
127:      */

```

```
128:     public String toString(){
129:         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+"";
130:     }
131: }
```

```

1: // MESSAGE HEARTBEAT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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).
11: */
12: public class msg_heartbeat extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_HEARTBEAT = 0;
15:     public static final int MAVLINK_MSG_LENGTH = 9;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_HEARTBEAT;
17:
18:     /**
19:     * A bitfield for use for autopilot-specific flags.
20:     */
21:     public int custom_mode;
22:     /**
23:     * Type of the MAV (quadrotor, helicopter, etc., up to 15 types, defined in
24:     MAV_TYPE ENUM)
25:     */
26:     public byte type;
27:     /**
28:     * Autopilot type / class. defined in MAV_AUTOPILOT ENUM
29:     */
30:     public byte autopilot;
31:     /**
32:     * System mode bitfield, see MAV_MODE_FLAGS ENUM in mavlink/include/mavlink
_types.h
33:     */
34:     public byte base_mode;
35:     /**
36:     * System status flag, see MAV_STATE ENUM
37:     */
38:     public byte system_status;
39:     /**
40:     * MAVLink version, not writable by user, gets added by protocol because of
magic data type: uint8_t_mavlink_version
41:     */
42:     public byte mavlink_version;
43:
44:     /**
45:     * Generates the payload for a mavlink message for a message of this type
46:     * @return
47:     */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_HEARTBEAT;
54:         packet.payload.putInt(custom_mode);
55:         packet.payload.putByte(type);
56:         packet.payload.putByte(autopilot);
57:         packet.payload.putByte(base_mode);
58:         packet.payload.putByte(system_status);
59:         packet.payload.putByte(mavlink_version);
60:         return packet;
61:     }
62:
63:     /**
64:     * Decode a heartbeat message into this class fields
65:     */
66:     * @param payload The message to decode
67:     */
68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         custom_mode = payload.getInt();
71:         type = payload.getByte();
72:         autopilot = payload.getByte();
73:         base_mode = payload.getByte();
74:         system_status = payload.getByte();
75:         mavlink_version = payload.getByte();
76:     }
77:
78:     /**
79:     * Constructor for a new message, just initializes the msgid
80:     */
81:     public msg_heartbeat(){
82:         msgid = MAVLINK_MSG_ID_HEARTBEAT;
83:     }
84:
85:     /**
86:     * Constructor for a new message, initializes the message with the payload
87:     * from a mavlink packet
88:     */
89:     public msg_heartbeat(MAVLinkPacket mavLinkPacket){
90:         this.sysid = mavLinkPacket.sysid;
91:         this.compid = mavLinkPacket.compid;
92:         this.msgid = MAVLINK_MSG_ID_HEARTBEAT;
93:         unpack(mavLinkPacket.payload);
94:         //Log.d("MAVLink", "HEARTBEAT");
95:         //Log.d("MAVLINK_MSG_ID_HEARTBEAT", toString());
96:     }
97:
98:
99:
100:     /**
101:     * Returns a string with the MSG name and data
102:     */
103:     public String toString(){
104:         return "MAVLINK_MSG_ID_HEARTBEAT -"+" custom_mode:"+custom_mode+" type:"+t
ype+" autopilot:"+autopilot+" base_mode:"+base_mode+" system_status:"+system_status+" mav
link_version:"+mavlink_version+"";
105:     }
106: }

```



```

1: // MESSAGE HIGHRES_IMU PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The IMU readings in SI units in NED body frame
11: */
12: public class msg_highres_imu extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_HIGHRES_IMU = 105;
15:     public static final int MAVLINK_MSG_LENGTH = 62;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_HIGHRES_IMU;
17:
18:
19:     /**
20:     * Timestamp (microseconds, synced to UNIX time or since system boot)
21:     */
22:     public long time_usec;
23:     /**
24:     * X acceleration (m/s^2)
25:     */
26:     public float xacc;
27:     /**
28:     * Y acceleration (m/s^2)
29:     */
30:     public float yacc;
31:     /**
32:     * Z acceleration (m/s^2)
33:     */
34:     public float zacc;
35:     /**
36:     * Angular speed around X axis (rad / sec)
37:     */
38:     public float xgyro;
39:     /**
40:     * Angular speed around Y axis (rad / sec)
41:     */
42:     public float ygyro;
43:     /**
44:     * Angular speed around Z axis (rad / sec)
45:     */
46:     public float zgyro;
47:     /**
48:     * X Magnetic field (Gauss)
49:     */
50:     public float xmag;
51:     /**
52:     * Y Magnetic field (Gauss)
53:     */
54:     public float ymag;
55:     /**
56:     * Z Magnetic field (Gauss)
57:     */
58:     public float zmag;
59:     /**
60:     * Absolute pressure in millibar
61:     */
62:     public float abs_pressure;
63:     /**
64:     * Differential pressure in millibar
65:     */
66:     public float diff_pressure;
67:     /**

```

```

68:     * Altitude calculated from pressure
69:     */
70:     public float pressure_alt;
71:     /**
72:     * Temperature in degrees celsius
73:     */
74:     public float temperature;
75:     /**
76:     * Bitmask for fields that have updated since last message, bit 0 = xacc, b
it 12: temperature
77:     */
78:     public short fields_updated;
79:
80:     /**
81:     * Generates the payload for a mavlink message for a message of this type
82:     * @return
83:     */
84:     public MAVLinkPacket pack(){
85:         MAVLinkPacket packet = new MAVLinkPacket();
86:         packet.len = MAVLINK_MSG_LENGTH;
87:         packet.sysid = 255;
88:         packet.compid = 190;
89:         packet.msgid = MAVLINK_MSG_ID_HIGHRES_IMU;
90:         packet.payload.putLong(time_usec);
91:         packet.payload.putFloat(xacc);
92:         packet.payload.putFloat(yacc);
93:         packet.payload.putFloat(zacc);
94:         packet.payload.putFloat(xgyro);
95:         packet.payload.putFloat(ygyro);
96:         packet.payload.putFloat(zgyro);
97:         packet.payload.putFloat(xmag);
98:         packet.payload.putFloat(ymag);
99:         packet.payload.putFloat(zmag);
100:         packet.payload.putFloat(abs_pressure);
101:         packet.payload.putFloat(diff_pressure);
102:         packet.payload.putFloat(pressure_alt);
103:         packet.payload.putFloat(temperature);
104:         packet.payload.putShort(fields_updated);
105:         return packet;
106:     }
107:
108:     /**
109:     * Decode a highres_imu message into this class fields
110:     *
111:     * @param payload The message to decode
112:     */
113:     public void unpack(MAVLinkPayload payload) {
114:         payload.resetIndex();
115:         time_usec = payload.getLong();
116:         xacc = payload.getFloat();
117:         yacc = payload.getFloat();
118:         zacc = payload.getFloat();
119:         xgyro = payload.getFloat();
120:         ygyro = payload.getFloat();
121:         zgyro = payload.getFloat();
122:         xmag = payload.getFloat();
123:         ymag = payload.getFloat();
124:         zmag = payload.getFloat();
125:         abs_pressure = payload.getFloat();
126:         diff_pressure = payload.getFloat();
127:         pressure_alt = payload.getFloat();
128:         temperature = payload.getFloat();
129:         fields_updated = payload.getShort();
130:     }
131:
132:     /**
133:     * Constructor for a new message, just initializes the msgid

```

```
134:     */
135:     public msg_highres_imu(){
136:         msgid = MAVLINK_MSG_ID_HIGHRES_IMU;
137:     }
138:
139:     /**
140:     * Constructor for a new message, initializes the message with the payload
141:     * from a mavlink packet
142:     *
143:     */
144:     public msg_highres_imu(MAVLinkPacket mavLinkPacket){
145:         this.sysid = mavLinkPacket.sysid;
146:         this.compid = mavLinkPacket.compid;
147:         this.msgid = MAVLINK_MSG_ID_HIGHRES_IMU;
148:         unpack(mavLinkPacket.payload);
149:         //Log.d("MAVLink", "HIGHRES_IMU");
150:         //Log.d("MAVLINK_MSG_ID_HIGHRES_IMU", toString());
151:     }
152:
153:
154:     /**
155:     * Returns a string with the MSG name and data
156:     */
157:     public String toString(){
158:         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+"";
159:     }
160: }
```

```

1: // MESSAGE HIL_CONTROLS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Sent from autopilot to simulation. Hardware in the loop control outputs
11:  */
12: public class msg_hil_controls extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_HIL_CONTROLS = 91;
15:     public static final int MAVLINK_MSG_LENGTH = 42;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_HIL_CONTROLS;
17:
18:
19:     /**
20:      * Timestamp (microseconds since UNIX epoch or microseconds since system bo
21:      ot)
22:      */
23:     public long time_usec;
24:     /**
25:      * Control output -1 .. 1
26:      */
27:     public float roll_ailerons;
28:     /**
29:      * Control output -1 .. 1
30:      */
31:     public float pitch_elevator;
32:     /**
33:      * Control output -1 .. 1
34:      */
35:     public float yaw_rudder;
36:     /**
37:      * Throttle 0 .. 1
38:      */
39:     public float throttle;
40:     /**
41:      * Aux 1, -1 .. 1
42:      */
43:     public float aux1;
44:     /**
45:      * Aux 2, -1 .. 1
46:      */
47:     public float aux2;
48:     /**
49:      * Aux 3, -1 .. 1
50:      */
51:     public float aux3;
52:     /**
53:      * Aux 4, -1 .. 1
54:      */
55:     public float aux4;
56:     /**
57:      * System mode (MAV_MODE)
58:      */
59:     public byte mode;
60:     /**
61:      * Navigation mode (MAV_NAV_MODE)
62:      */
63:     public byte nav_mode;
64:
65:     /**
66:      * Generates the payload for a mavlink message for a message of this type
67:      * @return

```

```

67:     */
68:     public MAVLinkPacket pack(){
69:         MAVLinkPacket packet = new MAVLinkPacket();
70:         packet.len = MAVLINK_MSG_LENGTH;
71:         packet.sysid = 255;
72:         packet.compid = 190;
73:         packet.msgid = MAVLINK_MSG_ID_HIL_CONTROLS;
74:         packet.payload.putLong(time_usec);
75:         packet.payload.putFloat(roll_ailerons);
76:         packet.payload.putFloat(pitch_elevator);
77:         packet.payload.putFloat(yaw_rudder);
78:         packet.payload.putFloat(throttle);
79:         packet.payload.putFloat(aux1);
80:         packet.payload.putFloat(aux2);
81:         packet.payload.putFloat(aux3);
82:         packet.payload.putFloat(aux4);
83:         packet.payload.putByte(mode);
84:         packet.payload.putByte(nav_mode);
85:         return packet;
86:     }
87:
88:     /**
89:      * Decode a hil_controls message into this class fields
90:      *
91:      * @param payload The message to decode
92:      */
93:     public void unpack(MAVLinkPayload payload) {
94:         payload.resetIndex();
95:         time_usec = payload.getLong();
96:         roll_ailerons = payload.getFloat();
97:         pitch_elevator = payload.getFloat();
98:         yaw_rudder = payload.getFloat();
99:         throttle = payload.getFloat();
100:         aux1 = payload.getFloat();
101:         aux2 = payload.getFloat();
102:         aux3 = payload.getFloat();
103:         aux4 = payload.getFloat();
104:         mode = payload.getByte();
105:         nav_mode = payload.getByte();
106:     }
107:
108:     /**
109:      * Constructor for a new message, just initializes the msgid
110:      */
111:     public msg_hil_controls(){
112:         msgid = MAVLINK_MSG_ID_HIL_CONTROLS;
113:     }
114:
115:     /**
116:      * Constructor for a new message, initializes the message with the payload
117:      * from a mavlink packet
118:      *
119:      */
120:     public msg_hil_controls(MAVLinkPacket mavLinkPacket){
121:         this.sysid = mavLinkPacket.sysid;
122:         this.compid = mavLinkPacket.compid;
123:         this.msgid = MAVLINK_MSG_ID_HIL_CONTROLS;
124:         unpack(mavLinkPacket.payload);
125:         //Log.d("MAVLink", "HIL_CONTROLS");
126:         //Log.d("MAVLINK_MSG_ID_HIL_CONTROLS", toString());
127:     }
128:
129:
130:     /**
131:      * Returns a string with the MSG name and data
132:      */
133:     public String toString(){

```

```
134:         return "MAVLINK_MSG_ID_HIL_CONTROLS -"+" time_usec:"+time_usec+" roll_aile  
rons:"+roll_ailerons+" pitch_elevator:"+pitch_elevator+" yaw_rudder:"+yaw_rudder+" throttl  
le:"+throttle+" aux1:"+aux1+" aux2:"+aux2+" aux3:"+aux3+" aux4:"+aux4+" mode:"+mode+" nav  
_mode:"+nav_mode+"";  
135:     }  
136: }
```

```

1: // MESSAGE HIL_RC_INPUTS_RAW PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Sent from simulation to autopilot. The RAW values of the RC channels received. The
11:  * standard PPM modulation is as follows: 1000 microseconds: 0%, 2000 microseconds: 100%.
12:  * Individual receivers/transmitters might violate this specification.
13:  */
14: public class msg_hil_rc_inputs_raw extends MAVLinkMessage{
15:
16:     public static final int MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW = 92;
17:     public static final int MAVLINK_MSG_LENGTH = 33;
18:     private static final long serialVersionUID = MAVLINK_MSG_ID_HIL_RC_INPUTS_
RAW;
19:
20:     /**
21:      * Timestamp (microseconds since UNIX epoch or microseconds since system bo
22:      * ot)
23:      */
24:     public long time_usec;
25:     /**
26:      * RC channel 1 value, in microseconds
27:      */
28:     public short chan1_raw;
29:     /**
30:      * RC channel 2 value, in microseconds
31:      */
32:     public short chan2_raw;
33:     /**
34:      * RC channel 3 value, in microseconds
35:      */
36:     public short chan3_raw;
37:     /**
38:      * RC channel 4 value, in microseconds
39:      */
40:     public short chan4_raw;
41:     /**
42:      * RC channel 5 value, in microseconds
43:      */
44:     public short chan5_raw;
45:     /**
46:      * RC channel 6 value, in microseconds
47:      */
48:     public short chan6_raw;
49:     /**
50:      * RC channel 7 value, in microseconds
51:      */
52:     public short chan7_raw;
53:     /**
54:      * RC channel 8 value, in microseconds
55:      */
56:     public short chan8_raw;
57:     /**
58:      * RC channel 9 value, in microseconds
59:      */
60:     public short chan9_raw;
61:     /**
62:      * RC channel 10 value, in microseconds
63:      */
63:     public short chan10_raw;
64:     /**

```

```

64:      * RC channel 11 value, in microseconds
65:      */
66:     public short chan11_raw;
67:     /**
68:      * RC channel 12 value, in microseconds
69:      */
70:     public short chan12_raw;
71:     /**
72:      * Receive signal strength indicator, 0: 0%, 255: 100%
73:      */
74:     public byte rssi;
75:
76:     /**
77:      * Generates the payload for a mavlink message for a message of this type
78:      * @return
79:      */
80:     public MAVLinkPacket pack(){
81:         MAVLinkPacket packet = new MAVLinkPacket();
82:         packet.len = MAVLINK_MSG_LENGTH;
83:         packet.sysid = 255;
84:         packet.compid = 190;
85:         packet.msgid = MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW;
86:         packet.payload.putLong(time_usec);
87:         packet.payload.putShort(chan1_raw);
88:         packet.payload.putShort(chan2_raw);
89:         packet.payload.putShort(chan3_raw);
90:         packet.payload.putShort(chan4_raw);
91:         packet.payload.putShort(chan5_raw);
92:         packet.payload.putShort(chan6_raw);
93:         packet.payload.putShort(chan7_raw);
94:         packet.payload.putShort(chan8_raw);
95:         packet.payload.putShort(chan9_raw);
96:         packet.payload.putShort(chan10_raw);
97:         packet.payload.putShort(chan11_raw);
98:         packet.payload.putShort(chan12_raw);
99:         packet.payload.putByte(rssi);
100:         return packet;
101:     }
102:
103:     /**
104:      * Decode a hil_rc_inputs_raw message into this class fields
105:      *
106:      * @param payload The message to decode
107:      */
108:     public void unpack(MAVLinkPayload payload) {
109:         payload.resetIndex();
110:         time_usec = payload.getLong();
111:         chan1_raw = payload.getShort();
112:         chan2_raw = payload.getShort();
113:         chan3_raw = payload.getShort();
114:         chan4_raw = payload.getShort();
115:         chan5_raw = payload.getShort();
116:         chan6_raw = payload.getShort();
117:         chan7_raw = payload.getShort();
118:         chan8_raw = payload.getShort();
119:         chan9_raw = payload.getShort();
120:         chan10_raw = payload.getShort();
121:         chan11_raw = payload.getShort();
122:         chan12_raw = payload.getShort();
123:         rssi = payload.getByte();
124:     }
125:
126:     /**
127:      * Constructor for a new message, just initializes the msgid
128:      */
129:     public msg_hil_rc_inputs_raw(){
130:         msgid = MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW;

```

```
131:     }
132:
133:     /**
134:      * Constructor for a new message, initializes the message with the payload
135:      * from a mavlink packet
136:      *
137:      */
138:     public msg_hil_rc_inputs_raw(MAVLinkPacket mavLinkPacket){
139:         this.sysid = mavLinkPacket.sysid;
140:         this.compid = mavLinkPacket.compid;
141:         this.msgid = MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW;
142:         unpack(mavLinkPacket.payload);
143:         //Log.d("MAVLink", "HIL_RC_INPUTS_RAW");
144:         //Log.d("MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW", toString());
145:     }
146:
147:
148:     /**
149:      * Returns a string with the MSG name and data
150:      */
151:     public String toString(){
152:         return "MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW -"+" time_usec:"+time_usec+" chan
1_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:"+ch
an8_raw+" chan9_raw:"+chan9_raw+" chan10_raw:"+chan10_raw+" chan11_raw:"+chan11_raw+" cha
n12_raw:"+chan12_raw+" rssi:"+rssi+"";
153:     }
154: }
```

```

1: // MESSAGE HIL_STATE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Sent from simulation to autopilot. This packet is useful for high throughput app
11:  * locations such as hardware in the loop simulations.
12:  */
13: public class msg_hil_state extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_HIL_STATE = 90;
16:     public static final int MAVLINK_MSG_LENGTH = 56;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_HIL_STATE;
18:
19:     /**
20:      * Timestamp (microseconds since UNIX epoch or microseconds since system bo
21:      * ot)
22:      */
23:     public long time_usec;
24:     /**
25:      * Roll angle (rad)
26:      */
27:     public float roll;
28:     /**
29:      * Pitch angle (rad)
30:      */
31:     public float pitch;
32:     /**
33:      * Yaw angle (rad)
34:      */
35:     public float yaw;
36:     /**
37:      * Roll angular speed (rad/s)
38:      */
39:     public float rollspeed;
40:     /**
41:      * Pitch angular speed (rad/s)
42:      */
43:     public float pitchspeed;
44:     /**
45:      * Yaw angular speed (rad/s)
46:      */
47:     public float yawspeed;
48:     /**
49:      * Latitude, expressed as * 1E7
50:      */
51:     public int lat;
52:     /**
53:      * Longitude, expressed as * 1E7
54:      */
55:     public int lon;
56:     /**
57:      * Altitude in meters, expressed as * 1000 (millimeters)
58:      */
59:     public int alt;
60:     /**
61:      * Ground X Speed (Latitude), expressed as m/s * 100
62:      */
63:     public short vx;
64:     /**
65:      * Ground Y Speed (Longitude), expressed as m/s * 100
66:      */

```

```

66:     public short vy;
67:     /**
68:      * Ground Z Speed (Altitude), expressed as m/s * 100
69:      */
70:     public short vz;
71:     /**
72:      * X acceleration (mg)
73:      */
74:     public short xacc;
75:     /**
76:      * Y acceleration (mg)
77:      */
78:     public short yacc;
79:     /**
80:      * Z acceleration (mg)
81:      */
82:     public short zacc;
83:
84:     /**
85:      * Generates the payload for a mavlink message for a message of this type
86:      * @return
87:      */
88:     public MAVLinkPacket pack(){
89:         MAVLinkPacket packet = new MAVLinkPacket();
90:         packet.len = MAVLINK_MSG_LENGTH;
91:         packet.sysid = 255;
92:         packet.compid = 190;
93:         packet.msgid = MAVLINK_MSG_ID_HIL_STATE;
94:         packet.payload.putLong(time_usec);
95:         packet.payload.putFloat(roll);
96:         packet.payload.putFloat(pitch);
97:         packet.payload.putFloat(yaw);
98:         packet.payload.putFloat(rollspeed);
99:         packet.payload.putFloat(pitchspeed);
100:        packet.payload.putFloat(yawspeed);
101:        packet.payload.putInt(lat);
102:        packet.payload.putInt(lon);
103:        packet.payload.putInt(alt);
104:        packet.payload.putShort(vx);
105:        packet.payload.putShort(vy);
106:        packet.payload.putShort(vz);
107:        packet.payload.putShort(xacc);
108:        packet.payload.putShort(yacc);
109:        packet.payload.putShort(zacc);
110:        return packet;
111:    }
112:
113:    /**
114:     * Decode a hil_state message into this class fields
115:     *
116:     * @param payload The message to decode
117:     */
118:    public void unpack(MAVLinkPayload payload) {
119:        payload.resetIndex();
120:        time_usec = payload.getLong();
121:        roll = payload.getFloat();
122:        pitch = payload.getFloat();
123:        yaw = payload.getFloat();
124:        rollspeed = payload.getFloat();
125:        pitchspeed = payload.getFloat();
126:        yawspeed = payload.getFloat();
127:        lat = payload.getInt();
128:        lon = payload.getInt();
129:        alt = payload.getInt();
130:        vx = payload.getShort();
131:        vy = payload.getShort();
132:        vz = payload.getShort();

```

```
133:         xacc = payload.getShort();
134:         yacc = payload.getShort();
135:         zacc = payload.getShort();
136:     }
137:
138:     /**
139:      * Constructor for a new message, just initializes the msgid
140:      */
141:     public msg_hil_state(){
142:         msgid = MAVLINK_MSG_ID_HIL_STATE;
143:     }
144:
145:     /**
146:      * Constructor for a new message, initializes the message with the payload
147:      * from a mavlink packet
148:      */
149:     public msg_hil_state(MAVLinkPacket mavLinkPacket){
150:         this.sysid = mavLinkPacket.sysid;
151:         this.compuid = mavLinkPacket.compuid;
152:         this.msgid = MAVLINK_MSG_ID_HIL_STATE;
153:         unpack(mavLinkPacket.payload);
154:         //Log.d("MAVLink", "HIL_STATE");
155:         //Log.d("MAVLINK_MSG_ID_HIL_STATE", toString());
156:     }
157:
158:
159:
160:     /**
161:      * Returns a string with the MSG name and data
162:      */
163:     public String toString(){
164:         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:" + xac
c + " yacc:" + yacc + " zacc:" + zacc + "";
165:     }
166: }
```



```

1: // MESSAGE HWSTATUS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Status of key hardware
11:  */
12: public class msg_hwstatus extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_HWSTATUS = 165;
15:     public static final int MAVLINK_MSG_LENGTH = 3;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_HWSTATUS;
17:
18:
19:     /**
20:      * board voltage (mV)
21:      */
22:     public short Vcc;
23:     /**
24:      * I2C error count
25:      */
26:     public byte I2Cerr;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compid = 190;
37:         packet.msgid = MAVLINK_MSG_ID_HWSTATUS;
38:         packet.payload.putShort(Vcc);
39:         packet.payload.putByte(I2Cerr);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a hwstatus message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         Vcc = payload.getShort();
51:         I2Cerr = payload.getByte();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_hwstatus(){
58:         msgid = MAVLINK_MSG_ID_HWSTATUS;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      *
65:      */
66:     public msg_hwstatus(MAVLinkPacket mavLinkPacket){
67:         this.sysid = mavLinkPacket.sysid;

```

```

68:         this.compid = mavLinkPacket.compid;
69:         this.msgid = MAVLINK_MSG_ID_HWSTATUS;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "HWSTATUS");
72:         //Log.d("MAVLINK_MSG_ID_HWSTATUS", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_HWSTATUS -"+" Vcc:"+Vcc+" I2Cerr:"+I2Cerr+"";
81:     }
82: }

```



```

1: // MESSAGE LIMITS_STATUS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Status of AP_Limits. Sent in extended
11:  * status stream when AP_Limits is enabled
12:  */
13: public class msg_limits_status extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_LIMITS_STATUS = 167;
16:     public static final int MAVLINK_MSG_LENGTH = 22;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_LIMITS_STATUS;
18:
19:
20:     /**
21:     * time of last breach in milliseconds since boot
22:     */
23:     public int last_trigger;
24:     /**
25:     * time of last recovery action in milliseconds since boot
26:     */
27:     public int last_action;
28:     /**
29:     * time of last successful recovery in milliseconds since boot
30:     */
31:     public int last_recovery;
32:     /**
33:     * time of last all-clear in milliseconds since boot
34:     */
35:     public int last_clear;
36:     /**
37:     * number of fence breaches
38:     */
39:     public short breach_count;
40:     /**
41:     * state of AP_Limits, (see enum LimitState, LIMITS_STATE)
42:     */
43:     public byte limits_state;
44:     /**
45:     * AP_Limit_Module bitfield of enabled modules, (see enum moduleid or LIMIT
_MODULE)
46:     */
47:     public byte mods_enabled;
48:     /**
49:     * AP_Limit_Module bitfield of required modules, (see enum moduleid or LIM
T_MODULE)
50:     */
51:     public byte mods_required;
52:     /**
53:     * AP_Limit_Module bitfield of triggered modules, (see enum moduleid or LIM
IT_MODULE)
54:     */
55:     public byte mods_triggered;
56:
57:     /**
58:     * Generates the payload for a mavlink message for a message of this type
59:     * @return
60:     */
61:     public MAVLinkPacket pack(){
62:         MAVLinkPacket packet = new MAVLinkPacket();
63:         packet.len = MAVLINK_MSG_LENGTH;
64:         packet.sysid = 255;

```

```

65:         packet.compid = 190;
66:         packet.msgid = MAVLINK_MSG_ID_LIMITS_STATUS;
67:         packet.payload.putInt(last_trigger);
68:         packet.payload.putInt(last_action);
69:         packet.payload.putInt(last_recovery);
70:         packet.payload.putInt(last_clear);
71:         packet.payload.putShort(breach_count);
72:         packet.payload.putByte(limits_state);
73:         packet.payload.putByte(mods_enabled);
74:         packet.payload.putByte(mods_required);
75:         packet.payload.putByte(mods_triggered);
76:         return packet;
77:     }
78:
79:     /**
80:     * Decode a limits_status message into this class fields
81:     *
82:     * @param payload The message to decode
83:     */
84:     public void unpack(MAVLinkPayload payload) {
85:         payload.resetIndex();
86:         last_trigger = payload.getInt();
87:         last_action = payload.getInt();
88:         last_recovery = payload.getInt();
89:         last_clear = payload.getInt();
90:         breach_count = payload.getShort();
91:         limits_state = payload.getByte();
92:         mods_enabled = payload.getByte();
93:         mods_required = payload.getByte();
94:         mods_triggered = payload.getByte();
95:     }
96:
97:     /**
98:     * Constructor for a new message, just initializes the msgid
99:     */
100:     public msg_limits_status(){
101:         msgid = MAVLINK_MSG_ID_LIMITS_STATUS;
102:     }
103:
104:     /**
105:     * Constructor for a new message, initializes the message with the payload
106:     * from a mavlink packet
107:     *
108:     */
109:     public msg_limits_status(MAVLinkPacket mavLinkPacket){
110:         this.sysid = mavLinkPacket.sysid;
111:         this.compid = mavLinkPacket.compid;
112:         this.msgid = MAVLINK_MSG_ID_LIMITS_STATUS;
113:         unpack(mavLinkPacket.payload);
114:         //Log.d("MAVLink", "LIMITS_STATUS");
115:         //Log.d("MAVLINK_MSG_ID_LIMITS_STATUS", toString());
116:     }
117:
118:
119:     /**
120:     * Returns a string with the MSG name and data
121:     */
122:     public String toString(){
123:         return "MAVLINK_MSG_ID_LIMITS_STATUS -"+" last_trigger:"+last_trigger+" la
st_action:"+last_action+" last_recovery:"+last_recovery+" last_clear:"+last_clear+" breac
h_count:"+breach_count+" limits_state:"+limits_state+" mods_enabled:"+mods_enabled+" mods
_required:"+mods_required+" mods_triggered:"+mods_triggered+"";
124:     }
125: }

```



```

1: // MESSAGE LOCAL_POSITION_NED PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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)
11:  */
12: public class msg_local_position_ned extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_LOCAL_POSITION_NED = 32;
15:     public static final int MAVLINK_MSG_LENGTH = 28;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_LOCAL_POSITION_NED;
17:
18:
19:     /**
20:      * Timestamp (milliseconds since system boot)
21:      */
22:     public int time_boot_ms;
23:     /**
24:      * X Position
25:      */
26:     public float x;
27:     /**
28:      * Y Position
29:      */
30:     public float y;
31:     /**
32:      * Z Position
33:      */
34:     public float z;
35:     /**
36:      * X Speed
37:      */
38:     public float vx;
39:     /**
40:      * Y Speed
41:      */
42:     public float vy;
43:     /**
44:      * Z Speed
45:      */
46:     public float vz;
47:
48:     /**
49:      * Generates the payload for a mavlink message for a message of this type
50:      * @return
51:      */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compuid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED;
58:         packet.payload.putInt(time_boot_ms);
59:         packet.payload.putFloat(x);
60:         packet.payload.putFloat(y);
61:         packet.payload.putFloat(z);
62:         packet.payload.putFloat(vx);
63:         packet.payload.putFloat(vy);
64:         packet.payload.putFloat(vz);

```

```

65:         return packet;
66:     }
67:
68:     /**
69:      * Decode a local_position_ned message into this class fields
70:      *
71:      * @param payload The message to decode
72:      */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         time_boot_ms = payload.getInt();
76:         x = payload.getFloat();
77:         y = payload.getFloat();
78:         z = payload.getFloat();
79:         vx = payload.getFloat();
80:         vy = payload.getFloat();
81:         vz = payload.getFloat();
82:     }
83:
84:     /**
85:      * Constructor for a new message, just initializes the msgid
86:      */
87:     public msg_local_position_ned(){
88:         msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED;
89:     }
90:
91:     /**
92:      * Constructor for a new message, initializes the message with the payload from a mavlink packet
93:      *
94:      */
95:     public msg_local_position_ned(MAVLinkPacket mavLinkPacket){
96:         this.sysid = mavLinkPacket.sysid;
97:         this.compuid = mavLinkPacket.compuid;
98:         this.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED;
99:         unpack(mavLinkPacket.payload);
100:         //Log.d("MAVLink", "LOCAL_POSITION_NED");
101:         //Log.d("MAVLINK_MSG_ID_LOCAL_POSITION_NED", toString());
102:     }
103:
104:
105:
106:     /**
107:      * Returns a string with the MSG name and data
108:      */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_LOCAL_POSITION_NED -" + " time_boot_ms:" + time_boot_ms
111:         + " x:" + x + " y:" + y + " z:" + z + " vx:" + vx + " vy:" + vy + " vz:" + vz + "";
112:     }

```



```

1: // MESSAGE LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The offset in X, Y, Z and yaw between the LOCAL_POSITION_NED messages of MAV X and
11: * the global coordinate frame in NED coordinates. Coordinate frame is right-handed, Z-axis
12: * is down (aeronautical frame, NED / north-east-down convention)
13: */
14: public class msg_local_position_ned_system_global_offset extends MAVLinkMessage {
15:     public static final int MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET = 89;
16:     public static final int MAVLINK_MSG_LENGTH = 28;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;
18:
19:     /**
20:      * Timestamp (milliseconds since system boot)
21:      */
22:     public int time_boot_ms;
23:     /**
24:      * X Position
25:      */
26:     public float x;
27:     /**
28:      * Y Position
29:      */
30:     public float y;
31:     /**
32:      * Z Position
33:      */
34:     public float z;
35:     /**
36:      * Roll
37:      */
38:     public float roll;
39:     /**
40:      * Pitch
41:      */
42:     public float pitch;
43:     /**
44:      * Yaw
45:      */
46:     public float yaw;
47:
48:     /**
49:      * Generates the payload for a mavlink message for a message of this type
50:      * @return
51:      */
52:     public MAVLinkPacket pack() {
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;
58:         packet.payload.putInt(time_boot_ms);
59:         packet.payload.putFloat(x);
60:         packet.payload.putFloat(y);
61:         packet.payload.putFloat(z);
62:         packet.payload.putFloat(roll);
63:         packet.payload.putFloat(pitch);
64:         packet.payload.putFloat(yaw);
65:         return packet;
66:     }
67:
68:     /**
69:      * Decode a local_position_ned_system_global_offset message into this class fields
70:      */
71:     /**
72:      * @param payload The message to decode
73:      */
74:     public void unpack(MAVLinkPayload payload) {
75:         payload.resetIndex();
76:         time_boot_ms = payload.getInt();
77:         x = payload.getFloat();
78:         y = payload.getFloat();
79:         z = payload.getFloat();
80:         roll = payload.getFloat();
81:         pitch = payload.getFloat();
82:         yaw = payload.getFloat();
83:     }
84:
85:     /**
86:      * Constructor for a new message, just initializes the msgid
87:      */
88:     public msg_local_position_ned_system_global_offset() {
89:         msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;
90:     }
91:
92:     /**
93:      * Constructor for a new message, initializes the message with the payload
94:      * from a mavlink packet
95:      */
96:     public msg_local_position_ned_system_global_offset(MAVLinkPacket mavLinkPacket) {
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compid = mavLinkPacket.compid;
99:         this.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET");
102:         //Log.d("MAVLINK_MSG_ID_LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET", toString());
103:     }
104:
105:     /**
106:      * Returns a string with the MSG name and data
107:      */
108:     public String toString() {
109:         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 + " ";
110:     }
111: }
112:

```



```

1: // MESSAGE LOCAL_POSITION_SETPOINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Transmit the current local setpoint of the controller to other MAVs (collision a
11:  * voidance) and to the GCS.
12:  */
13: public class msg_local_position_setpoint extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT = 51;
16:     public static final int MAVLINK_MSG_LENGTH = 17;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_LOCAL_POSITION
18: _SETPOINT;
19:
20:     /**
21:     * x position
22:     */
23:     public float x;
24:     /**
25:     * y position
26:     */
27:     public float y;
28:     /**
29:     * z position
30:     */
31:     public float z;
32:     /**
33:     * Desired yaw angle
34:     */
35:     public float yaw;
36:     /**
37:     * Coordinate frame - valid values are only MAV_FRAME_LOCAL_NED or MAV_FRAM
38: E_LOCAL_ENU
39:     */
40:     public byte coordinate_frame;
41:
42:     /**
43:     * Generates the payload for a mavlink message for a message of this type
44:     * @return
45:     */
46:     public MAVLinkPacket pack(){
47:         MAVLinkPacket packet = new MAVLinkPacket();
48:         packet.len = MAVLINK_MSG_LENGTH;
49:         packet.sysid = 255;
50:         packet.compid = 190;
51:         packet.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT;
52:         packet.payload.putFloat(x);
53:         packet.payload.putFloat(y);
54:         packet.payload.putFloat(z);
55:         packet.payload.putFloat(yaw);
56:         packet.payload.putByte(coordinate_frame);
57:         return packet;
58:     }
59:
60:     /**
61:     * Decode a local_position_setpoint message into this class fields
62:     */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:
66:         x = payload.getFloat();
67:         y = payload.getFloat();
68:         z = payload.getFloat();
69:         yaw = payload.getFloat();
70:         coordinate_frame = payload.getByte();
71:     }
72:
73:     /**
74:     * Constructor for a new message, just initializes the msgid
75:     */
76:     public msg_local_position_setpoint(){
77:         msgid = MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT;
78:     }
79:
80:     /**
81:     * Constructor for a new message, initializes the message with the payload
82:     * from a mavlink packet
83:     */
84:     public msg_local_position_setpoint(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compid = mavLinkPacket.compid;
87:         this.msgid = MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "LOCAL_POSITION_SETPOINT");
90:         //Log.d("MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT", toString());
91:     }
92:
93:     /**
94:     * Returns a string with the MSG name and data
95:     */
96:     public String toString(){
97:         return "MAVLINK_MSG_ID_LOCAL_POSITION_SETPOINT -"+" x:"+x+" y:"+y+" z:"+z+
98: " yaw:"+yaw+" coordinate_frame:"+coordinate_frame+"";
99:     }
100: }

```



```

1: // MESSAGE MANUAL_CONTROL PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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
11: */
12: public class msg_manual_control extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MANUAL_CONTROL = 69;
15:     public static final int MAVLINK_MSG_LENGTH = 11;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MANUAL_CONTROL
;
17:
18:
19:     /**
20:      * X-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indic
  ates that this axis is invalid. Generally corresponds to forward(1000)-backward(-1000) mo
  vement on a joystick and the pitch of a vehicle.
21:      */
22:     public short x;
23:     /**
24:      * Y-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indic
  ates that this axis is invalid. Generally corresponds to left(-1000)-right(1000) movement
  on a joystick and the roll of a vehicle.
25:      */
26:     public short y;
27:     /**
28:      * Z-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indic
  ates 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.
29:      */
30:     public short z;
31:     /**
32:      * R-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indic
  ates 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.
33:      */
34:     public short r;
35:     /**
36:      * A bitfield corresponding to the joystick buttons' current state, 1 for p
  ressed, 0 for released. The lowest bit corresponds to Button 1.
37:      */
38:     public short buttons;
39:     /**
40:      * The system to be controlled.
41:      */
42:     public byte target;
43:
44:     /**
45:      * Generates the payload for a mavlink message for a message of this type
46:      * @return
47:      */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compuid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_MANUAL_CONTROL;
54:         packet.payload.putShort(x);
55:         packet.payload.putShort(y);

```

```

56:         packet.payload.putShort(z);
57:         packet.payload.putShort(r);
58:         packet.payload.putShort(buttons);
59:         packet.payload.putByte(target);
60:         return packet;
61:     }
62:
63:     /**
64:      * Decode a manual_control message into this class fields
65:      *
66:      * @param payload The message to decode
67:      */
68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         x = payload.getShort();
71:         y = payload.getShort();
72:         z = payload.getShort();
73:         r = payload.getShort();
74:         buttons = payload.getShort();
75:         target = payload.getByte();
76:     }
77:
78:     /**
79:      * Constructor for a new message, just initializes the msgid
80:      */
81:     public msg_manual_control(){
82:         msgid = MAVLINK_MSG_ID_MANUAL_CONTROL;
83:     }
84:
85:     /**
86:      * Constructor for a new message, initializes the message with the payload
87:      * from a mavlink packet
88:      *
89:      */
90:     public msg_manual_control(MAVLinkPacket mavLinkPacket){
91:         this.sysid = mavLinkPacket.sysid;
92:         this.compuid = mavLinkPacket.compuid;
93:         this.msgid = MAVLINK_MSG_ID_MANUAL_CONTROL;
94:         unpack(mavLinkPacket.payload);
95:         //Log.d("MAVLink", "MANUAL_CONTROL");
96:         //Log.d("MAVLINK_MSG_ID_MANUAL_CONTROL", toString());
97:     }
98:
99:
100:     /**
101:      * Returns a string with the MSG name and data
102:      */
103:     public String toString(){
104:         return "MAVLINK_MSG_ID_MANUAL_CONTROL -"+" x:"+x+" y:"+y+" z:"+z+" r:"+r+"
  buttons:"+buttons+" target:"+target+"";
105:     }
106: }

```



```

1: // MESSAGE MANUAL_SETPOINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Setpoint in roll, pitch, yaw and thrust from the operator
11:  */
12: public class msg_manual_setpoint extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MANUAL_SETPOINT = 81;
15:     public static final int MAVLINK_MSG_LENGTH = 22;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MANUAL_SETPOIN
T;
17:
18:
19:     /**
20:      * Timestamp in milliseconds since system boot
21:      */
22:     public int time_boot_ms;
23:     /**
24:      * Desired roll rate in radians per second
25:      */
26:     public float roll;
27:     /**
28:      * Desired pitch rate in radians per second
29:      */
30:     public float pitch;
31:     /**
32:      * Desired yaw rate in radians per second
33:      */
34:     public float yaw;
35:     /**
36:      * Collective thrust, normalized to 0 .. 1
37:      */
38:     public float thrust;
39:     /**
40:      * Flight mode switch position, 0.. 255
41:      */
42:     public byte mode_switch;
43:     /**
44:      * Override mode switch position, 0.. 255
45:      */
46:     public byte manual_override_switch;
47:
48:     /**
49:      * Generates the payload for a mavlink message for a message of this type
50:      * @return
51:      */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_MANUAL_SETPOINT;
58:         packet.payload.putInt(time_boot_ms);
59:         packet.payload.putFloat(roll);
60:         packet.payload.putFloat(pitch);
61:         packet.payload.putFloat(yaw);
62:         packet.payload.putFloat(thrust);
63:         packet.payload.putByte(mode_switch);
64:         packet.payload.putByte(manual_override_switch);
65:         return packet;
66:     }

```

```

67:
68:     /**
69:      * Decode a manual_setpoint message into this class fields
70:      */
71:     * @param payload The message to decode
72:     */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         time_boot_ms = payload.getInt();
76:         roll = payload.getFloat();
77:         pitch = payload.getFloat();
78:         yaw = payload.getFloat();
79:         thrust = payload.getFloat();
80:         mode_switch = payload.getByte();
81:         manual_override_switch = payload.getByte();
82:     }
83:
84:     /**
85:      * Constructor for a new message, just initializes the msgid
86:      */
87:     public msg_manual_setpoint(){
88:         msgid = MAVLINK_MSG_ID_MANUAL_SETPOINT;
89:     }
90:
91:     /**
92:      * Constructor for a new message, initializes the message with the payload
93:      * from a mavlink packet
94:      */
95:     *
96:     public msg_manual_setpoint(MAVLinkPacket mavLinkPacket){
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compid = mavLinkPacket.compid;
99:         this.msgid = MAVLINK_MSG_ID_MANUAL_SETPOINT;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "MANUAL_SETPOINT");
102:         //Log.d("MAVLINK_MSG_ID_MANUAL_SETPOINT", toString());
103:     }
104:
105:
106:     /**
107:      * Returns a string with the MSG name and data
108:      */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_MANUAL_SETPOINT -"+" time_boot_ms:"+time_boot_ms+"
roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+" thrust:"+thrust+" mode_switch:"+mode_switch+" m
anual_override_switch:"+manual_override_switch+"";
111:     }
112: }

```



```

1: // MESSAGE MEMINFO PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * state of APM memory
11:  */
12: public class msg_meminfo extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MEMINFO = 152;
15:     public static final int MAVLINK_MSG_LENGTH = 4;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MEMINFO;
17:
18:
19:     /**
20:      * heap top
21:      */
22:     public short brkval;
23:     /**
24:      * free memory
25:      */
26:     public short freemem;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compid = 190;
37:         packet.msgid = MAVLINK_MSG_ID_MEMINFO;
38:         packet.payload.putShort(brkval);
39:         packet.payload.putShort(freemem);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a meminfo message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         brkval = payload.getShort();
51:         freemem = payload.getShort();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_meminfo(){
58:         msgid = MAVLINK_MSG_ID_MEMINFO;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      *
65:      */
66:     public msg_meminfo(MAVLinkPacket mavLinkPacket){
67:         this.sysid = mavLinkPacket.sysid;

```

```

68:         this.compid = mavLinkPacket.compid;
69:         this.msgid = MAVLINK_MSG_ID_MEMINFO;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "MEMINFO");
72:         //Log.d("MAVLINK_MSG_ID_MEMINFO", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_MEMINFO -"+" brkval:"+brkval+" freemem:"+freemem+"";
81:     }
82: }

```



```

1: // MESSAGE MEMORY_VECT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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.
11:  */
12: public class msg_memory_vect extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MEMORY_VECT = 249;
15:     public static final int MAVLINK_MSG_LENGTH = 36;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MEMORY_VECT;
17:
18:     /**
19:     * Starting address of the debug variables
20:     */
21:     public short address;
22:     /**
23:     * Version code of the type variable. 0=unknown, type ignored and assumed i
24:     nt16_t. 1=as below
25:     */
26:     public byte ver;
27:     /**
28:     * Type code of the memory variables. for ver = 1: 0=16 x int16_t, 1=16 x u
29:     int16_t, 2=16 x Q15, 3=16 x IQ14
30:     */
31:     public byte type;
32:     /**
33:     * Memory contents at specified address
34:     */
35:     public byte value[] = new byte[32];
36:
37:     /**
38:     * Generates the payload for a mavlink message for a message of this type
39:     * @return
40:     */
41:     public MAVLinkPacket pack(){
42:         MAVLinkPacket packet = new MAVLinkPacket();
43:         packet.len = MAVLINK_MSG_LENGTH;
44:         packet.sysid = 255;
45:         packet.compid = 190;
46:         packet.msgid = MAVLINK_MSG_ID_MEMORY_VECT;
47:         packet.payload.putShort(address);
48:         packet.payload.putByte(ver);
49:         packet.payload.putByte(type);
50:         for (int i = 0; i < value.length; i++) {
51:             packet.payload.putByte(value[i]);
52:         }
53:         return packet;
54:     }
55:
56:     /**
57:     * Decode a memory_vect message into this class fields
58:     *
59:     * @param payload The message to decode
60:     */
61:     public void unpack(MAVLinkPayload payload) {
62:         payload.resetIndex();
63:         address = payload.getShort();
64:         ver = payload.getByte();

```

```

64:         type = payload.getByte();
65:         for (int i = 0; i < value.length; i++) {
66:             value[i] = payload.getByte();
67:         }
68:     }
69:
70:     /**
71:     * Constructor for a new message, just initializes the msgid
72:     */
73:     public msg_memory_vect(){
74:         msgid = MAVLINK_MSG_ID_MEMORY_VECT;
75:     }
76:
77:     /**
78:     * Constructor for a new message, initializes the message with the payload
79:     * from a mavlink packet
80:     */
81:     public msg_memory_vect(MAVLinkPacket mavLinkPacket){
82:         this.sysid = mavLinkPacket.sysid;
83:         this.compid = mavLinkPacket.compid;
84:         this.msgid = MAVLINK_MSG_ID_MEMORY_VECT;
85:         unpack(mavLinkPacket.payload);
86:         //Log.d("MAVLink", "MEMORY_VECT");
87:         //Log.d("MAVLINK_MSG_ID_MEMORY_VECT", toString());
88:     }
89:
90:
91:
92:     /**
93:     * Returns a string with the MSG name and data
94:     */
95:     public String toString(){
96:         return "MAVLINK_MSG_ID_MEMORY_VECT -"+" address:"+address+" ver:"+ver+" ty
97: pe:"+type+" value:"+value+"";
98:     }

```



```

1: // MESSAGE MISSION_CLEAR_ALL PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Delete all mission items at once.
11:  */
12: public class msg_mission_clear_all extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_CLEAR_ALL = 45;
15:     public static final int MAVLINK_MSG_LENGTH = 2;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_CLEAR_
ALL;
17:
18:
19:     /**
20:      * System ID
21:      */
22:     public byte target_system;
23:     /**
24:      * Component ID
25:      */
26:     public byte target_component;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compId = 190;
37:         packet.msgid = MAVLINK_MSG_ID_MISSION_CLEAR_ALL;
38:         packet.payload.putByte(target_system);
39:         packet.payload.putByte(target_component);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a mission_clear_all message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         target_system = payload.getByte();
51:         target_component = payload.getByte();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_mission_clear_all(){
58:         msgid = MAVLINK_MSG_ID_MISSION_CLEAR_ALL;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      *
65:      */
66:     public msg_mission_clear_all(MAVLinkPacket mavLinkPacket){

```

```

67:         this.sysid = mavLinkPacket.sysid;
68:         this.compId = mavLinkPacket.compId;
69:         this.msgid = MAVLINK_MSG_ID_MISSION_CLEAR_ALL;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "MISSION_CLEAR_ALL");
72:         //Log.d("MAVLINK_MSG_ID_MISSION_CLEAR_ALL", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_MISSION_CLEAR_ALL -" + target_system:" +target_syste
m+" target_component:" +target_component+"";
81:     }
82: }

```



```

1: // MESSAGE MISSION_COUNT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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.
11:  */
12: public class msg_mission_count extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_COUNT = 44;
15:     public static final int MAVLINK_MSG_LENGTH = 4;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_COUNT;
17:
18:     /**
19:      * Number of mission items in the sequence
20:      */
21:     public short count;
22:     /**
23:      * System ID
24:      */
25:     public byte target_system;
26:     /**
27:      * Component ID
28:      */
29:     public byte target_component;
30:
31:     /**
32:      * Generates the payload for a mavlink message for a message of this type
33:      * @return
34:      */
35:     public MAVLinkPacket pack(){
36:         MAVLinkPacket packet = new MAVLinkPacket();
37:         packet.len = MAVLINK_MSG_LENGTH;
38:         packet.sysid = 255;
39:         packet.compuid = 190;
40:         packet.msgid = MAVLINK_MSG_ID_MISSION_COUNT;
41:         packet.payload.putShort(count);
42:         packet.payload.putByte(target_system);
43:         packet.payload.putByte(target_component);
44:         return packet;
45:     }
46:
47:     /**
48:      * Decode a mission_count message into this class fields
49:      *
50:      * @param payload The message to decode
51:      */
52:     public void unpack(MAVLinkPayload payload) {
53:         payload.resetIndex();
54:         count = payload.getShort();
55:         target_system = payload.getByte();
56:         target_component = payload.getByte();
57:     }
58:
59:     /**
60:      * Constructor for a new message, just initializes the msgid
61:      */
62:     public msg_mission_count(){
63:         msgid = MAVLINK_MSG_ID_MISSION_COUNT;
64:     }
65:

```

```

66:
67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     public msg_mission_count(MAVLinkPacket mavLinkPacket){
72:         this.sysid = mavLinkPacket.sysid;
73:         this.compuid = mavLinkPacket.compuid;
74:         this.msgid = MAVLINK_MSG_ID_MISSION_COUNT;
75:         unpack(mavLinkPacket.payload);
76:         //Log.d("MAVLink", "MISSION_COUNT");
77:         //Log.d("MAVLINK_MSG_ID_MISSION_COUNT", toString());
78:     }
79:
80:
81:     /**
82:      * Returns a string with the MSG name and data
83:      */
84:     public String toString(){
85:         return "MAVLINK_MSG_ID_MISSION_COUNT -"+" count:"+count+" target_system:"+target_system+" target_component:"+target_component+"";
86:     }
87:
88: }

```



```

1: // MESSAGE MISSION_CURRENT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Message that announces the sequence number of the current active mission item. The
11:  * MAV will fly towards this mission item.
12:  */
13: public class msg_mission_current extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_MISSION_CURRENT = 42;
16:     public static final int MAVLINK_MSG_LENGTH = 2;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_CURRENT;
18:
19:     /**
20:      * Sequence
21:      */
22:     public short seq;
23:
24:     /**
25:      * Generates the payload for a mavlink message for a message of this type
26:      * @return
27:      */
28:     public MAVLinkPacket pack(){
29:         MAVLinkPacket packet = new MAVLinkPacket();
30:         packet.len = MAVLINK_MSG_LENGTH;
31:         packet.sysid = 255;
32:         packet.compid = 190;
33:         packet.msgid = MAVLINK_MSG_ID_MISSION_CURRENT;
34:         packet.payload.putShort(seq);
35:         return packet;
36:     }
37:
38:     /**
39:      * Decode a mission_current message into this class fields
40:      *
41:      * @param payload The message to decode
42:      */
43:     public void unpack(MAVLinkPayload payload) {
44:         payload.resetIndex();
45:         seq = payload.getShort();
46:     }
47:
48:     /**
49:      * Constructor for a new message, just initializes the msgid
50:      */
51:     public msg_mission_current(){
52:         msgid = MAVLINK_MSG_ID_MISSION_CURRENT;
53:     }
54:
55:     /**
56:      * Constructor for a new message, initializes the message with the payload
57:      * from a mavlink packet
58:      */
59:     public msg_mission_current(MAVLinkPacket mavLinkPacket){
60:         this.sysid = mavLinkPacket.sysid;
61:         this.compid = mavLinkPacket.compid;
62:         this.msgid = MAVLINK_MSG_ID_MISSION_CURRENT;
63:         unpack(mavLinkPacket.payload);
64:         //Log.d("MAVLink", "MISSION_CURRENT");
65:

```

```

66:         //Log.d("MAVLINK_MSG_ID_MISSION_CURRENT", toString());
67:     }
68:
69:
70:     /**
71:      * Returns a string with the MSG name and data
72:      */
73:     public String toString(){
74:         return "MAVLINK_MSG_ID_MISSION_CURRENT -"+" seq:"+seq+"";
75:     }
76: }

```



```

1: // MESSAGE MISSION_ITEM PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Message encoding a mission item. This message is emitted to announce
11:  * the presence of a mission item and to set a mission item on the sy
12:  * stem. The mission item can be either in x, y, z meters (type: LOCAL) or x:lat, y:lon, z:a
13:  * ltitude. Local frame is Z-down, right handed (NED), global frame is Z-up, right handed (E
14:  * NU). See also http://qgroundcontrol.org/mavlink/waypoint_protocol.
15:  */
16: public class msg_mission_item extends MAVLinkMessage{
17:
18:     public static final int MAVLINK_MSG_ID_MISSION_ITEM = 39;
19:     public static final int MAVLINK_MSG_LENGTH = 37;
20:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_ITEM;
21:
22:     /**
23:      * PARAM1 / For NAV command MISSIONs: Radius in which the MISSION is accept
24:      * ed as reached, in meters
25:      */
26:     public float param1;
27:     /**
28:      * PARAM2 / For NAV command MISSIONs: Time that the MAV should stay inside
29:      * the PARAM1 radius before advancing, in milliseconds
30:      */
31:     public float param2;
32:     /**
33:      * PARAM3 / For LOITER command MISSIONs: Orbit to circle around the MISSION
34:      * , in meters. If positive the orbit direction should be clockwise, if negative the orbit d
35:      * irection should be counter-clockwise.
36:      */
37:     public float param3;
38:     /**
39:      * PARAM4 / For NAV and LOITER command MISSIONs: Yaw orientation in degrees
40:      * , [0..360] 0 = NORTH
41:      */
42:     public float param4;
43:     /**
44:      * PARAM5 / local: x position, global: latitude
45:      */
46:     public float x;
47:     /**
48:      * PARAM6 / y position: global: longitude
49:      */
50:     public float y;
51:     /**
52:      * PARAM7 / z position: global: altitude
53:      */
54:     public float z;
55:     /**
56:      * Sequence
57:      */
58:     public short seq;
59:     /**
60:      * The scheduled action for the MISSION. see MAV_CMD in common.xml MAVLink
61:      * specs
62:      */
63:     public short command;
64:     /**
65:      * System ID
66:      */

```

```

59:     public byte target_system;
60:     /**
61:      * Component ID
62:      */
63:     public byte target_component;
64:     /**
65:      * The coordinate system of the MISSION. see MAV_FRAME in mavlink_types.h
66:      */
67:     public byte frame;
68:     /**
69:      * false:0, true:1
70:      */
71:     public byte current;
72:     /**
73:      * autocontinue to next wp
74:      */
75:     public byte autocontinue;
76:
77:     /**
78:      * Generates the payload for a mavlink message for a message of this type
79:      * @return
80:      */
81:     public MAVLinkPacket pack(){
82:         MAVLinkPacket packet = new MAVLinkPacket();
83:         packet.len = MAVLINK_MSG_LENGTH;
84:         packet.sysid = 255;
85:         packet.compid = 190;
86:         packet.msgid = MAVLINK_MSG_ID_MISSION_ITEM;
87:         packet.payload.putFloat(param1);
88:         packet.payload.putFloat(param2);
89:         packet.payload.putFloat(param3);
90:         packet.payload.putFloat(param4);
91:         packet.payload.putFloat(x);
92:         packet.payload.putFloat(y);
93:         packet.payload.putFloat(z);
94:         packet.payload.putShort(seq);
95:         packet.payload.putShort(command);
96:         packet.payload.putByte(target_system);
97:         packet.payload.putByte(target_component);
98:         packet.payload.putByte(frame);
99:         packet.payload.putByte(current);
100:         packet.payload.putByte(autocontinue);
101:         return packet;
102:     }
103:
104:     /**
105:      * Decode a mission_item message into this class fields
106:      *
107:      * @param payload The message to decode
108:      */
109:     public void unpack(MAVLinkPayload payload) {
110:         payload.resetIndex();
111:         param1 = payload.getFloat();
112:         param2 = payload.getFloat();
113:         param3 = payload.getFloat();
114:         param4 = payload.getFloat();
115:         x = payload.getFloat();
116:         y = payload.getFloat();
117:         z = payload.getFloat();
118:         seq = payload.getShort();
119:         command = payload.getShort();
120:         target_system = payload.getByte();
121:         target_component = payload.getByte();
122:         frame = payload.getByte();
123:         current = payload.getByte();
124:         autocontinue = payload.getByte();
125:     }

```

```
126:
127:     /**
128:      * Constructor for a new message, just initializes the msgid
129:      */
130:     public msg_mission_item(){
131:         msgid = MAVLINK_MSG_ID_MISSION_ITEM;
132:     }
133:
134:     /**
135:      * Constructor for a new message, initializes the message with the payload
136:      * from a mavlink packet
137:      */
138:     public msg_mission_item(MAVLinkPacket mavLinkPacket){
139:         this.sysid = mavLinkPacket.sysid;
140:         this.compid = mavLinkPacket.compid;
141:         this.msgid = MAVLINK_MSG_ID_MISSION_ITEM;
142:         unpack(mavLinkPacket.payload);
143:         //Log.d("MAVLink", "MISSION_ITEM");
144:         //Log.d("MAVLINK_MSG_ID_MISSION_ITEM", toString());
145:     }
146:
147:
148:
149:     /**
150:      * Returns a string with the MSG name and data
151:      */
152:     public String toString(){
153:         return "MAVLINK_MSG_ID_MISSION_ITEM -"+" param1:"+param1+" param2:"+param2
154:         +" param3:"+param3+" param4:"+param4+" x:"+x+" y:"+y+" z:"+z+" seq:"+seq+" command:"+comm
155:         and+" target_system:"+target_system+" target_component:"+target_component+" frame:"+frame
156:         +" current:"+current+" autocontinue:"+autocontinue+"";
157:     }
158: }
```

```

1: // MESSAGE MISSION_ITEM_REACHED PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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.
11:  */
12: public class msg_mission_item_reached extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_ITEM_REACHED = 46;
15:     public static final int MAVLINK_MSG_LENGTH = 2;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;
17:
18:
19:     /**
20:      * Sequence
21:      */
22:     public short seq;
23:
24:     /**
25:      * Generates the payload for a mavlink message for a message of this type
26:      * @return
27:      */
28:     public MAVLinkPacket pack(){
29:         MAVLinkPacket packet = new MAVLinkPacket();
30:         packet.len = MAVLINK_MSG_LENGTH;
31:         packet.sysid = 255;
32:         packet.compid = 190;
33:         packet.msgid = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;
34:         packet.payload.putShort(seq);
35:         return packet;
36:     }
37:
38:     /**
39:      * Decode a mission_item_reached message into this class fields
40:      *
41:      * @param payload The message to decode
42:      */
43:     public void unpack(MAVLinkPayload payload) {
44:         payload.resetIndex();
45:         seq = payload.getShort();
46:     }
47:
48:     /**
49:      * Constructor for a new message, just initializes the msgid
50:      */
51:     public msg_mission_item_reached(){
52:         msgid = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;
53:     }
54:
55:     /**
56:      * Constructor for a new message, initializes the message with the payload
57:      * from a mavlink packet
58:      *
59:      */
60:     public msg_mission_item_reached(MAVLinkPacket mavLinkPacket){
61:         this.sysid = mavLinkPacket.sysid;
62:         this.compid = mavLinkPacket.compid;
63:         this.msgid = MAVLINK_MSG_ID_MISSION_ITEM_REACHED;
64:         unpack(mavLinkPacket.payload);

```

```

65:         //Log.d("MAVLink", "MISSION_ITEM_REACHED");
66:         //Log.d("MAVLINK_MSG_ID_MISSION_ITEM_REACHED", toString());
67:     }
68:
69:
70:     /**
71:      * Returns a string with the MSG name and data
72:      */
73:     public String toString(){
74:         return "MAVLINK_MSG_ID_MISSION_ITEM_REACHED -"+" seq:"+seq+"";
75:     }
76: }

```



```

1: // MESSAGE MISSION_REQUEST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Request the information of the mission item with the sequence number seq. The re
    sponse of the system to this message should be a MISSION_ITEM message. http://qgroundcont
    rol.org/mavlink/waypoint_protocol
11: */
12: public class msg_mission_request extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_REQUEST = 40;
15:     public static final int MAVLINK_MSG_LENGTH = 4;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_REQUES
T;
17:
18:
19:     /**
20:      * Sequence
21:      */
22:     public short seq;
23:     /**
24:      * System ID
25:      */
26:     public byte target_system;
27:     /**
28:      * Component ID
29:      */
30:     public byte target_component;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_MISSION_REQUEST;
42:         packet.payload.putShort(seq);
43:         packet.payload.putByte(target_system);
44:         packet.payload.putByte(target_component);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a mission_request message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         seq = payload.getShort();
56:         target_system = payload.getByte();
57:         target_component = payload.getByte();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_mission_request(){
64:         msgid = MAVLINK_MSG_ID_MISSION_REQUEST;

```

```

65:     }
66:
67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     /**
72:      * public msg_mission_request(MAVLinkPacket mavLinkPacket){
73:         this.sysid = mavLinkPacket.sysid;
74:         this.compid = mavLinkPacket.compid;
75:         this.msgid = MAVLINK_MSG_ID_MISSION_REQUEST;
76:         unpack(mavLinkPacket.payload);
77:         //Log.d("MAVLink", "MISSION_REQUEST");
78:         //Log.d("MAVLINK_MSG_ID_MISSION_REQUEST", toString());
79:     }
80:
81:
82:     /**
83:      * Returns a string with the MSG name and data
84:      */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_MISSION_REQUEST -"+" seq:"+seq+" target_system:"+ta
rget_system+" target_component:"+target_component+"";
87:     }
88: }

```



```

1: // MESSAGE MISSION_REQUEST_LIST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Request the overall list of mission items from the system/component.
11:  */
12: public class msg_mission_request_list extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_REQUEST_LIST = 43;
15:     public static final int MAVLINK_MSG_LENGTH = 2;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_REQUEST_LIST;
17:
18:
19:     /**
20:      * System ID
21:      */
22:     public byte target_system;
23:     /**
24:      * Component ID
25:      */
26:     public byte target_component;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compid = 190;
37:         packet.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_LIST;
38:         packet.payload.putByte(target_system);
39:         packet.payload.putByte(target_component);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a mission_request_list message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         target_system = payload.getByte();
51:         target_component = payload.getByte();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_mission_request_list(){
58:         msgid = MAVLINK_MSG_ID_MISSION_REQUEST_LIST;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      */
65:
66:     public msg_mission_request_list(MAVLinkPacket mavLinkPacket){
67:         this.sysid = mavLinkPacket.sysid;
68:         this.compid = mavLinkPacket.compid;
69:         this.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_LIST;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "MISSION_REQUEST_LIST");
72:         //Log.d("MAVLINK_MSG_ID_MISSION_REQUEST_LIST", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_MISSION_REQUEST_LIST -" + target_system + "target_system+" + target_component + "target_component";
81:     }
82: }

```



```

1: // MESSAGE MISSION_REQUEST_PARTIAL_LIST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Request a partial list of mission items from the system/component. http://qgroun
dcontrol.org/mavlink/waypoint_protocol. If start and end index are the same, just send on
e waypoint.
11: */
12: public class msg_mission_request_partial_list extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST = 37;
15:     public static final int MAVLINK_MSG_LENGTH = 6;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_REQUES
T_PARTIAL_LIST;
17:
18:
19:     /**
20:      * Start index, 0 by default
21:      */
22:     public short start_index;
23:     /**
24:      * End index, -1 by default (-1: send list to end). Else a valid index of t
he list
25:      */
26:     public short end_index;
27:     /**
28:      * System ID
29:      */
30:     public byte target_system;
31:     /**
32:      * Component ID
33:      */
34:     public byte target_component;
35:
36:     /**
37:      * Generates the payload for a mavlink message for a message of this type
38:      * @return
39:      */
40:     public MAVLinkPacket pack(){
41:         MAVLinkPacket packet = new MAVLinkPacket();
42:         packet.len = MAVLINK_MSG_LENGTH;
43:         packet.sysid = 255;
44:         packet.compid = 190;
45:         packet.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST;
46:         packet.payload.putShort(start_index);
47:         packet.payload.putShort(end_index);
48:         packet.payload.putByte(target_system);
49:         packet.payload.putByte(target_component);
50:         return packet;
51:     }
52:
53:     /**
54:      * Decode a mission_request_partial_list message into this class fields
55:      *
56:      * @param payload The message to decode
57:      */
58:     public void unpack(MAVLinkPayload payload) {
59:         payload.resetIndex();
60:         start_index = payload.getShort();
61:         end_index = payload.getShort();
62:         target_system = payload.getByte();
63:         target_component = payload.getByte();
64:     }
65:
66:     /**
67:      * Constructor for a new message, just initializes the msgid
68:      */
69:     public msg_mission_request_partial_list(){
70:         msgid = MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST;
71:     }
72:
73:     /**
74:      * Constructor for a new message, initializes the message with the payload
75:      * from a mavlink packet
76:      */
77:     public msg_mission_request_partial_list(MAVLinkPacket mavLinkPacket){
78:         this.sysid = mavLinkPacket.sysid;
79:         this.compid = mavLinkPacket.compid;
80:         this.msgid = MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST;
81:         unpack(mavLinkPacket.payload);
82:         //Log.d("MAVLink", "MISSION_REQUEST_PARTIAL_LIST");
83:         //Log.d("MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST", toString());
84:     }
85:
86:
87:
88:     /**
89:      * Returns a string with the MSG name and data
90:      */
91:     public String toString(){
92:         return "MAVLINK_MSG_ID_MISSION_REQUEST_PARTIAL_LIST -"+" start_index:"+sta
rt_index+" end_index:"+end_index+" target_system:"+target_system+" target_component:"+tar
get_component+"";
93:     }
94: }

```



```

1: // MESSAGE MISSION_SET_CURRENT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set the mission item with sequence number seq as current item. This means that t
he MAV will continue to this mission item on the shortest path (not following the mission
items in-between).
11: */
12: public class msg_mission_set_current extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_SET_CURRENT = 41;
15:     public static final int MAVLINK_MSG_LENGTH = 4;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_SET_CU
RRENT;
17:
18:
19:     /**
20:      * Sequence
21:      */
22:     public short seq;
23:     /**
24:      * System ID
25:      */
26:     public byte target_system;
27:     /**
28:      * Component ID
29:      */
30:     public byte target_component;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_MISSION_SET_CURRENT;
42:         packet.payload.putShort(seq);
43:         packet.payload.putByte(target_system);
44:         packet.payload.putByte(target_component);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a mission_set_current message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         seq = payload.getShort();
56:         target_system = payload.getByte();
57:         target_component = payload.getByte();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_mission_set_current(){
64:         msgid = MAVLINK_MSG_ID_MISSION_SET_CURRENT;

```

```

65:     }
66:
67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     /**
72:     public msg_mission_set_current(MAVLinkPacket mavLinkPacket){
73:         this.sysid = mavLinkPacket.sysid;
74:         this.compid = mavLinkPacket.compid;
75:         this.msgid = MAVLINK_MSG_ID_MISSION_SET_CURRENT;
76:         unpack(mavLinkPacket.payload);
77:         //Log.d("MAVLink", "MISSION_SET_CURRENT");
78:         //Log.d("MAVLINK_MSG_ID_MISSION_SET_CURRENT", toString());
79:     }
80:
81:
82:     /**
83:      * Returns a string with the MSG name and data
84:      */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_MISSION_SET_CURRENT -"+" seq:"+seq+" target_system:
"+target_system+" target_component:"+target_component+"";
87:     }
88: }

```



```

1: // MESSAGE MISSION_WRITE_PARTIAL_LIST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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!
11:  */
12: public class msg_mission_write_partial_list extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST = 38;
15:     public static final int MAVLINK_MSG_LENGTH = 6;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;
17:
18:     /**
19:      * Start index, 0 by default and smaller / equal to the largest index of the current onboard list.
20:      */
21:     public short start_index;
22:     /**
23:      * End index, equal or greater than start index.
24:      */
25:     public short end_index;
26:     /**
27:      * System ID
28:      */
29:     public byte target_system;
30:     /**
31:      * Component ID
32:      */
33:     public byte target_component;
34:
35:     /**
36:      * Generates the payload for a mavlink message for a message of this type
37:      * @return
38:      */
39:     public MAVLinkPacket pack(){
40:         MAVLinkPacket packet = new MAVLinkPacket();
41:         packet.len = MAVLINK_MSG_LENGTH;
42:         packet.sysid = 255;
43:         packet.compid = 190;
44:         packet.msgid = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;
45:         packet.payload.putShort(start_index);
46:         packet.payload.putShort(end_index);
47:         packet.payload.putByte(target_system);
48:         packet.payload.putByte(target_component);
49:         return packet;
50:     }
51:
52:     /**
53:      * Decode a mission_write_partial_list message into this class fields
54:      *
55:      * @param payload The message to decode
56:      */
57:     public void unpack(MAVLinkPayload payload) {
58:         payload.resetIndex();
59:         start_index = payload.getShort();
60:         end_index = payload.getShort();
61:         target_system = payload.getByte();
62:         target_component = payload.getByte();
63:
64:     }
65:
66:     /**
67:      * Constructor for a new message, just initializes the msgid
68:      */
69:     public msg_mission_write_partial_list(){
70:         msgid = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;
71:     }
72:
73:     /**
74:      * Constructor for a new message, initializes the message with the payload
75:      * from a mavlink packet
76:      */
77:     public msg_mission_write_partial_list(MAVLinkPacket mavLinkPacket){
78:         this.sysid = mavLinkPacket.sysid;
79:         this.compid = mavLinkPacket.compid;
80:         this.msgid = MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST;
81:         unpack(mavLinkPacket.payload);
82:         //Log.d("MAVLink", "MISSION_WRITE_PARTIAL_LIST");
83:         //Log.d("MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST", toString());
84:     }
85:
86:     /**
87:      * Returns a string with the MSG name and data
88:      */
89:     public String toString(){
90:         return "MAVLINK_MSG_ID_MISSION_WRITE_PARTIAL_LIST -" + start_index + " end_index:" + end_index + " target_system:" + target_system + " target_component:" + target_component + "";
91:     }
92:
93: }
94:

```



```

1: // MESSAGE MOUNT_CONFIGURE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Message to configure a camera mount, directional antenna, etc.
11:  */
12: public class msg_mount_configure extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MOUNT_CONFIGURE = 156;
15:     public static final int MAVLINK_MSG_LENGTH = 6;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MOUNT_CONFIGUR
E;
17:
18:
19:     /**
20:      * System ID
21:      */
22:     public byte target_system;
23:     /**
24:      * Component ID
25:      */
26:     public byte target_component;
27:     /**
28:      * mount operating mode (see MAV_MOUNT_MODE enum)
29:      */
30:     public byte mount_mode;
31:     /**
32:      * (1 = yes, 0 = no)
33:      */
34:     public byte stab_roll;
35:     /**
36:      * (1 = yes, 0 = no)
37:      */
38:     public byte stab_pitch;
39:     /**
40:      * (1 = yes, 0 = no)
41:      */
42:     public byte stab_yaw;
43:
44:     /**
45:      * Generates the payload for a mavlink message for a message of this type
46:      * @return
47:      */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_MOUNT_CONFIGURE;
54:         packet.payload.putByte(target_system);
55:         packet.payload.putByte(target_component);
56:         packet.payload.putByte(mount_mode);
57:         packet.payload.putByte(stab_roll);
58:         packet.payload.putByte(stab_pitch);
59:         packet.payload.putByte(stab_yaw);
60:         return packet;
61:     }
62:
63:     /**
64:      * Decode a mount_configure message into this class fields
65:      *
66:      * @param payload The message to decode

```

```

67:     */
68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         target_system = payload.getByte();
71:         target_component = payload.getByte();
72:         mount_mode = payload.getByte();
73:         stab_roll = payload.getByte();
74:         stab_pitch = payload.getByte();
75:         stab_yaw = payload.getByte();
76:     }
77:
78:     /**
79:      * Constructor for a new message, just initializes the msgid
80:      */
81:     public msg_mount_configure(){
82:         msgid = MAVLINK_MSG_ID_MOUNT_CONFIGURE;
83:     }
84:
85:     /**
86:      * Constructor for a new message, initializes the message with the payload
87:      * from a mavlink packet
88:      */
89:     public msg_mount_configure(MAVLinkPacket mavLinkPacket){
90:         this.sysid = mavLinkPacket.sysid;
91:         this.compid = mavLinkPacket.compid;
92:         this.msgid = MAVLINK_MSG_ID_MOUNT_CONFIGURE;
93:         unpack(mavLinkPacket.payload);
94:         //Log.d("MAVLink", "MOUNT_CONFIGURE");
95:         //Log.d("MAVLINK_MSG_ID_MOUNT_CONFIGURE", toString());
96:     }
97:
98:
99:
100:     /**
101:      * Returns a string with the MSG name and data
102:      */
103:     public String toString(){
104:         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+"";
105:     }
106: }

```



```

1: // MESSAGE MOUNT_CONTROL PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Message to control a camera mount, directional antenna, etc.
11:  */
12: public class msg_mount_control extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MOUNT_CONTROL = 157;
15:     public static final int MAVLINK_MSG_LENGTH = 15;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MOUNT_CONTROL;
17:
18:
19:     /**
20:      * pitch(deg*100) or lat, depending on mount mode
21:      */
22:     public int input_a;
23:     /**
24:      * roll(deg*100) or lon depending on mount mode
25:      */
26:     public int input_b;
27:     /**
28:      * yaw(deg*100) or alt (in cm) depending on mount mode
29:      */
30:     public int input_c;
31:     /**
32:      * System ID
33:      */
34:     public byte target_system;
35:     /**
36:      * Component ID
37:      */
38:     public byte target_component;
39:     /**
40:      * if "1" it will save current trimmed position on EEPROM (just valid for N
41:      * EUTRAL and LANDING)
42:      */
43:     public byte save_position;
44:
45:     /**
46:      * Generates the payload for a mavlink message for a message of this type
47:      * @return
48:      */
49:     public MAVLinkPacket pack(){
50:         MAVLinkPacket packet = new MAVLinkPacket();
51:         packet.len = MAVLINK_MSG_LENGTH;
52:         packet.sysid = 255;
53:         packet.compid = 190;
54:         packet.msgid = MAVLINK_MSG_ID_MOUNT_CONTROL;
55:         packet.payload.putInt(input_a);
56:         packet.payload.putInt(input_b);
57:         packet.payload.putInt(input_c);
58:         packet.payload.putByte(target_system);
59:         packet.payload.putByte(target_component);
60:         packet.payload.putByte(save_position);
61:         return packet;
62:     }
63:
64:     /**
65:      * Decode a mount_control message into this class fields
66:      * @param payload The message to decode

```

```

67:     */
68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         input_a = payload.getInt();
71:         input_b = payload.getInt();
72:         input_c = payload.getInt();
73:         target_system = payload.getByte();
74:         target_component = payload.getByte();
75:         save_position = payload.getByte();
76:     }
77:
78:     /**
79:      * Constructor for a new message, just initializes the msgid
80:      */
81:     public msg_mount_control(){
82:         msgid = MAVLINK_MSG_ID_MOUNT_CONTROL;
83:     }
84:
85:     /**
86:      * Constructor for a new message, initializes the message with the payload
87:      * from a mavlink packet
88:      */
89:     public msg_mount_control(MAVLinkPacket mavLinkPacket){
90:         this.sysid = mavLinkPacket.sysid;
91:         this.compid = mavLinkPacket.compid;
92:         this.msgid = MAVLINK_MSG_ID_MOUNT_CONTROL;
93:         unpack(mavLinkPacket.payload);
94:         //Log.d("MAVLink", "MOUNT_CONTROL");
95:         //Log.d("MAVLINK_MSG_ID_MOUNT_CONTROL", toString());
96:     }
97:
98:
99:
100:     /**
101:      * Returns a string with the MSG name and data
102:      */
103:     public String toString(){
104:         return "MAVLINK_MSG_ID_MOUNT_CONTROL -"+" input_a:"+input_a+" input_b:"+in
105:         put_b+" input_c:"+input_c+" target_system:"+target_system+" target_component:"+target_com
106:         ponent+" save_position:"+save_position+"";

```



```

1: // MESSAGE MOUNT_STATUS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Message with some status from APM to GCS about camera or antenna mount
11:  */
12: public class msg_mount_status extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MOUNT_STATUS = 158;
15:     public static final int MAVLINK_MSG_LENGTH = 14;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MOUNT_STATUS;
17:
18:
19:     /**
20:      * pitch(deg*100) or lat, depending on mount mode
21:      */
22:     public int pointing_a;
23:     /**
24:      * roll(deg*100) or lon depending on mount mode
25:      */
26:     public int pointing_b;
27:     /**
28:      * yaw(deg*100) or alt (in cm) depending on mount mode
29:      */
30:     public int pointing_c;
31:     /**
32:      * System ID
33:      */
34:     public byte target_system;
35:     /**
36:      * Component ID
37:      */
38:     public byte target_component;
39:
40:     /**
41:      * Generates the payload for a mavlink message for a message of this type
42:      * @return
43:      */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_MOUNT_STATUS;
50:         packet.payload.putInt(pointing_a);
51:         packet.payload.putInt(pointing_b);
52:         packet.payload.putInt(pointing_c);
53:         packet.payload.putByte(target_system);
54:         packet.payload.putByte(target_component);
55:         return packet;
56:     }
57:
58:     /**
59:      * Decode a mount_status message into this class fields
60:      *
61:      * @param payload The message to decode
62:      */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         pointing_a = payload.getInt();
66:         pointing_b = payload.getInt();
67:         pointing_c = payload.getInt();

```

```

68:         target_system = payload.getByte();
69:         target_component = payload.getByte();
70:     }
71:
72:     /**
73:      * Constructor for a new message, just initializes the msgid
74:      */
75:     public msg_mount_status(){
76:         msgid = MAVLINK_MSG_ID_MOUNT_STATUS;
77:     }
78:
79:     /**
80:      * Constructor for a new message, initializes the message with the payload
81:      * from a mavlink packet
82:      */
83:     /**
84:     public msg_mount_status(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compid = mavLinkPacket.compid;
87:         this.msgid = MAVLINK_MSG_ID_MOUNT_STATUS;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "MOUNT_STATUS");
90:         //Log.d("MAVLINK_MSG_ID_MOUNT_STATUS", toString());
91:     }
92:
93:
94:     /**
95:      * Returns a string with the MSG name and data
96:      */
97:     public String toString(){
98:         return "MAVLINK_MSG_ID_MOUNT_STATUS -"+" pointing_a:"+pointing_a+" pointin
99:         g_b:"+pointing_b+" pointing_c:"+pointing_c+" target_system:"+target_system+" target_compo
100:         nent:"+target_component+"";
101:     }

```


./com/MAVLink/Messages/ardupilotmega/msg_named_value_float.java

```
1: // MESSAGE NAMED_VALUE_FLOAT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Send a key-value pair as float. The use of this message is discouraged for norma
11:  * l packets, but a quite efficient way for testing new messages and getting experimental de
12:  * bug output.
13:  */
14: public class msg_named_value_float extends MAVLinkMessage{
15:
16:     public static final int MAVLINK_MSG_ID_NAMED_VALUE_FLOAT = 251;
17:     public static final int MAVLINK_MSG_LENGTH = 18;
18:     private static final long serialVersionUID = MAVLINK_MSG_ID_NAMED_VALUE_FL
19:
20: OAT;
21:
22:     /**
23:     * Timestamp (milliseconds since system boot)
24:     */
25:     public int time_boot_ms;
26:     /**
27:     * Floating point value
28:     */
29:     public float value;
30:     /**
31:     * Name of the debug variable
32:     */
33:     public byte name[] = new byte[10];
34:
35:     /**
36:     * Generates the payload for a mavlink message for a message of this type
37:     * @return
38:     */
39:     public MAVLinkPacket pack(){
40:         MAVLinkPacket packet = new MAVLinkPacket();
41:         packet.len = MAVLINK_MSG_LENGTH;
42:         packet.sysid = 255;
43:         packet.compid = 190;
44:         packet.msgid = MAVLINK_MSG_ID_NAMED_VALUE_FLOAT;
45:         packet.payload.putInt(time_boot_ms);
46:         packet.payload.putFloat(value);
47:         for (int i = 0; i < name.length; i++) {
48:             packet.payload.putByte(name[i]);
49:         }
50:         return packet;
51:     }
52:
53:     /**
54:     * Decode a named_value_float message into this class fields
55:     *
56:     * @param payload The message to decode
57:     */
58:     public void unpack(MAVLinkPayload payload) {
59:         payload.resetIndex();
60:         time_boot_ms = payload.getInt();
61:         value = payload.getFloat();
62:         for (int i = 0; i < name.length; i++) {
63:             name[i] = payload.getByte();
64:         }
65:     }
66:
67:     /**
```

Fri Oct 25 14:10:51 2013

1

```
65:     * Constructor for a new message, just initializes the msgid
66:     */
67:     public msg_named_value_float(){
68:         msgid = MAVLINK_MSG_ID_NAMED_VALUE_FLOAT;
69:     }
70:
71:     /**
72:     * Constructor for a new message, initializes the message with the payload
73:     * from a mavlink packet
74:     */
75:     public msg_named_value_float(MAVLinkPacket mavLinkPacket){
76:         this.sysid = mavLinkPacket.sysid;
77:         this.compid = mavLinkPacket.compid;
78:         this.msgid = MAVLINK_MSG_ID_NAMED_VALUE_FLOAT;
79:         unpack(mavLinkPacket.payload);
80:         //Log.d("MAVLink", "NAMED_VALUE_FLOAT");
81:         //Log.d("MAVLINK_MSG_ID_NAMED_VALUE_FLOAT", toString());
82:     }
83:
84:     /**
85:     * Sets the buffer of this message with a string, adds the necessary padding
86:     */
87:     public void setName(String str) {
88:         int len = Math.min(str.length(), 10);
89:         for (int i=0; i<len; i++) {
90:             name[i] = (byte) str.charAt(i);
91:         }
92:         for (int i=len; i<10; i++) { // padding for the rest of
93: the buffer
94:             name[i] = 0;
95:         }
96:     }
97:
98:     /**
99:     * Gets the message, formatted as a string
100:     */
101:     public String getName() {
102:         String result = "";
103:         for (int i = 0; i < 10; i++) {
104:             if (name[i] != 0)
105:                 result = result + (char) name[i];
106:             else
107:                 break;
108:         }
109:         return result;
110:     }
111:
112:     /**
113:     * Returns a string with the MSG name and data
114:     */
115:     public String toString(){
116:         return "MAVLINK_MSG_ID_NAMED_VALUE_FLOAT -"+" time_boot_ms:"+time_boot_ms+
117: " value:"+value+" name:"+name+"";
118:     }
119: }
```



```

1: // MESSAGE DEBUG PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Send a debug value. The index is used to discriminate between values. These values
11:  * show up in the plot of QGroundControl as DEBUG N.
12:  */
13: public class msg_debug extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_DEBUG = 254;
16:     public static final int MAVLINK_MSG_LENGTH = 9;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_DEBUG;
18:
19:     /**
20:      * Timestamp (milliseconds since system boot)
21:      */
22:     public int time_boot_ms;
23:     /**
24:      * DEBUG value
25:      */
26:     public float value;
27:     /**
28:      * index of debug variable
29:      */
30:     public byte ind;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_DEBUG;
42:         packet.payload.putInt(time_boot_ms);
43:         packet.payload.putFloat(value);
44:         packet.payload.putByte(ind);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a debug message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         time_boot_ms = payload.getInt();
56:         value = payload.getFloat();
57:         ind = payload.getByte();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_debug(){
64:         msgid = MAVLINK_MSG_ID_DEBUG;
65:     }
66:

```

```

67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     public msg_debug(MAVLinkPacket mavLinkPacket){
72:         this.sysid = mavLinkPacket.sysid;
73:         this.compid = mavLinkPacket.compid;
74:         this.msgid = MAVLINK_MSG_ID_DEBUG;
75:         unpack(mavLinkPacket.payload);
76:         //Log.d("MAVLink", "DEBUG");
77:         //Log.d("MAVLINK_MSG_ID_DEBUG", toString());
78:     }
79:
80:
81:
82:     /**
83:      * Returns a string with the MSG name and data
84:      */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_DEBUG -"+" time_boot_ms:"+time_boot_ms+" value:"+value+" ind:"+ind+"";
87:     }
88: }

```



```

1: // MESSAGE GPS_STATUS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * The positioning status, as reported by GPS. This message is intended to display
11:  * status information about each satellite visible to the receiver. See message GLOBAL_POSIT
12:  * ION for the global position estimate. This message can contain information for up to 20 s
13:  * atellites.
14:  */
15: public class msg_gps_status extends MAVLinkMessage{
16:
17:     public static final int MAVLINK_MSG_ID_GPS_STATUS = 25;
18:     public static final int MAVLINK_MSG_LENGTH = 101;
19:     private static final long serialVersionUID = MAVLINK_MSG_ID_GPS_STATUS;
20:
21:     /**
22:      * Number of satellites visible
23:      */
24:     public byte satellites_visible;
25:     /**
26:      * Global satellite ID
27:      */
28:     public byte satellite_prn[] = new byte[20];
29:     /**
30:      * 0: Satellite not used, 1: used for localization
31:      */
32:     public byte satellite_used[] = new byte[20];
33:     /**
34:      * Elevation (0: right on top of receiver, 90: on the horizon) of satellite
35:      */
36:     public byte satellite_elevation[] = new byte[20];
37:     /**
38:      * Direction of satellite, 0: 0 deg, 255: 360 deg.
39:      */
40:     public byte satellite_azimuth[] = new byte[20];
41:     /**
42:      * Signal to noise ratio of satellite
43:      */
44:     public byte satellite_snr[] = new byte[20];
45:     /**
46:      * Generates the payload for a mavlink message for a message of this type
47:      * @return
48:      */
49:     public MAVLinkPacket pack(){
50:         MAVLinkPacket packet = new MAVLinkPacket();
51:         packet.len = MAVLINK_MSG_LENGTH;
52:         packet.sysid = 255;
53:         packet.compuid = 190;
54:         packet.msgid = MAVLINK_MSG_ID_GPS_STATUS;
55:         packet.payload.putByte(satellites_visible);
56:         for (int i = 0; i < satellite_prn.length; i++) {
57:             packet.payload.putByte(satellite_prn[i]);
58:         }
59:         for (int i = 0; i < satellite_used.length; i++) {
60:             packet.payload.putByte(satellite_used[i]);
61:         }
62:         for (int i = 0; i < satellite_elevation.length; i++) {
63:             packet.payload.putByte(satellite_elevation[i]);
64:         }
65:         for (int i = 0; i < satellite_azimuth.length; i++) {

```

```

66:             packet.payload.putByte(satellite_azimuth[i]);
67:         }
68:         for (int i = 0; i < satellite_snr.length; i++) {
69:             packet.payload.putByte(satellite_snr[i]);
70:         }
71:         return packet;
72:     }
73:
74:     /**
75:      * Decode a gps_status message into this class fields
76:      *
77:      * @param payload The message to decode
78:      */
79:     public void unpack(MAVLinkPayload payload) {
80:         payload.resetIndex();
81:         satellites_visible = payload.getByte();
82:         for (int i = 0; i < satellite_prn.length; i++) {
83:             satellite_prn[i] = payload.getByte();
84:         }
85:         for (int i = 0; i < satellite_used.length; i++) {
86:             satellite_used[i] = payload.getByte();
87:         }
88:         for (int i = 0; i < satellite_elevation.length; i++) {
89:             satellite_elevation[i] = payload.getByte();
90:         }
91:         for (int i = 0; i < satellite_azimuth.length; i++) {
92:             satellite_azimuth[i] = payload.getByte();
93:         }
94:         for (int i = 0; i < satellite_snr.length; i++) {
95:             satellite_snr[i] = payload.getByte();
96:         }
97:     }
98:
99:     /**
100:      * Constructor for a new message, just initializes the msgid
101:      */
102:     public msg_gps_status(){
103:         msgid = MAVLINK_MSG_ID_GPS_STATUS;
104:     }
105:
106:     /**
107:      * Constructor for a new message, initializes the message with the payload
108:      * from a mavlink packet
109:      */
110:     public msg_gps_status(MAVLinkPacket mavLinkPacket){
111:         this.sysid = mavLinkPacket.sysid;
112:         this.compuid = mavLinkPacket.compuid;
113:         this.msgid = MAVLINK_MSG_ID_GPS_STATUS;
114:         unpack(mavLinkPacket.payload);
115:         //Log.d("MAVLink", "GPS_STATUS");
116:         //Log.d("MAVLINK_MSG_ID_GPS_STATUS", toString());
117:     }
118:
119:     /**
120:      * Returns a string with the MSG name and data
121:      */
122:     public String toString(){
123:         return "MAVLINK_MSG_ID_GPS_STATUS -"+" satellites_visible:"+satellites_vis
124:         ible+" satellite_prn:"+satellite_prn+" satellite_used:"+satellite_used+" satellite_elevat
125:         ion:"+satellite_elevation+" satellite_azimuth:"+satellite_azimuth+" satellite_snr:"+satel
126:         lite_snr+"";
127:     }

```



```

1: // MESSAGE MISSION_ACK PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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).
11:  */
12: public class msg_mission_ack extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_MISSION_ACK = 47;
15:     public static final int MAVLINK_MSG_LENGTH = 3;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_MISSION_ACK;
17:
18:
19:     /**
20:      * System ID
21:      */
22:     public byte target_system;
23:     /**
24:      * Component ID
25:      */
26:     public byte target_component;
27:     /**
28:      * See MAV_MISSION_RESULT enum
29:      */
30:     public byte type;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_MISSION_ACK;
42:         packet.payload.putByte(target_system);
43:         packet.payload.putByte(target_component);
44:         packet.payload.putByte(type);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a mission_ack message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         target_system = payload.getByte();
56:         target_component = payload.getByte();
57:         type = payload.getByte();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_mission_ack(){
64:         msgid = MAVLINK_MSG_ID_MISSION_ACK;
65:     }
66:

```

```

67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     public msg_mission_ack(MAVLinkPacket mavLinkPacket){
72:         this.sysid = mavLinkPacket.sysid;
73:         this.compid = mavLinkPacket.compid;
74:         this.msgid = MAVLINK_MSG_ID_MISSION_ACK;
75:         unpack(mavLinkPacket.payload);
76:         //Log.d("MAVLink", "MISSION_ACK");
77:         //Log.d("MAVLINK_MSG_ID_MISSION_ACK", toString());
78:     }
79:
80:
81:
82:     /**
83:      * Returns a string with the MSG name and data
84:      */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_MISSION_ACK -"+" target_system:"+target_system+" target_component:"+target_component+" type:"+type+"";
87:     }
88: }

```



```

1: // MESSAGE NAMED_VALUE_INT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * 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.
11: */
12: public class msg_named_value_int extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_NAMED_VALUE_INT = 252;
15:     public static final int MAVLINK_MSG_LENGTH = 18;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_NAMED_VALUE_INT;
17:
18:
19:     /**
20:     * Timestamp (milliseconds since system boot)
21:     */
22:     public int time_boot_ms;
23:     /**
24:     * Signed integer value
25:     */
26:     public int value;
27:     /**
28:     * Name of the debug variable
29:     */
30:     public byte name[] = new byte[10];
31:
32:     /**
33:     * Generates the payload for a mavlink message for a message of this type
34:     * @return
35:     */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_NAMED_VALUE_INT;
42:         packet.payload.putInt(time_boot_ms);
43:         packet.payload.putInt(value);
44:         for (int i = 0; i < name.length; i++) {
45:             packet.payload.putByte(name[i]);
46:         }
47:         return packet;
48:     }
49:
50:     /**
51:     * Decode a named_value_int message into this class fields
52:     *
53:     * @param payload The message to decode
54:     */
55:     public void unpack(MAVLinkPayload payload) {
56:         payload.resetIndex();
57:         time_boot_ms = payload.getInt();
58:         value = payload.getInt();
59:         for (int i = 0; i < name.length; i++) {
60:             name[i] = payload.getByte();
61:         }
62:     }
63:
64:     /**

```

```

65:     * Constructor for a new message, just initializes the msgid
66:     */
67:     public msg_named_value_int(){
68:         msgid = MAVLINK_MSG_ID_NAMED_VALUE_INT;
69:     }
70:
71:     /**
72:     * Constructor for a new message, initializes the message with the payload
73:     * from a mavlink packet
74:     */
75:     public msg_named_value_int(MAVLinkPacket mavLinkPacket){
76:         this.sysid = mavLinkPacket.sysid;
77:         this.compid = mavLinkPacket.compid;
78:         this.msgid = MAVLINK_MSG_ID_NAMED_VALUE_INT;
79:         unpack(mavLinkPacket.payload);
80:         //Log.d("MAVLink", "NAMED_VALUE_INT");
81:         //Log.d("MAVLINK_MSG_ID_NAMED_VALUE_INT", toString());
82:     }
83:
84:
85:     /**
86:     * Sets the buffer of this message with a string, adds the necessary padding
87:     */
88:     public void setName(String str) {
89:         int len = Math.min(str.length(), 10);
90:         for (int i=0; i<len; i++) {
91:             name[i] = (byte) str.charAt(i);
92:         }
93:         for (int i=len; i<10; i++) { // padding for the rest of
the buffer
94:             name[i] = 0;
95:         }
96:     }
97:
98:     /**
99:     * Gets the message, formatted as a string
100:     */
101:     public String getName() {
102:         String result = "";
103:         for (int i = 0; i < 10; i++) {
104:             if (name[i] != 0)
105:                 result = result + (char) name[i];
106:             else
107:                 break;
108:         }
109:         return result;
110:     }
111:
112:     /**
113:     * Returns a string with the MSG name and data
114:     */
115:     public String toString(){
116:         return "MAVLINK_MSG_ID_NAMED_VALUE_INT -"+" time_boot_ms:"+time_boot_ms+"
value:"+value+" name:"+name+"";
117:     }
118: }

```



```

1: // MESSAGE ROLL_PITCH_YAW_RATES_THRUST_SETPOINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Setpoint in roll, pitch, yaw rates and thrust currently active on the system.
11:  */
12: public class msg_roll_pitch_yaw_rates_thrust_setpoint extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT
T = 80;
15:     public static final int MAVLINK_MSG_LENGTH = 20;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_ROLL_PITCH_YAW
_RATES_THRUST_SETPOINT;
17:
18:
19:     /**
20:     * Timestamp in milliseconds since system boot
21:     */
22:     public int time_boot_ms;
23:     /**
24:     * Desired roll rate in radians per second
25:     */
26:     public float roll_rate;
27:     /**
28:     * Desired pitch rate in radians per second
29:     */
30:     public float pitch_rate;
31:     /**
32:     * Desired yaw rate in radians per second
33:     */
34:     public float yaw_rate;
35:     /**
36:     * Collective thrust, normalized to 0 .. 1
37:     */
38:     public float thrust;
39:
40:     /**
41:     * Generates the payload for a mavlink message for a message of this type
42:     * @return
43:     */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compuid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT
;
50:
51:         packet.payload.putInt(time_boot_ms);
52:         packet.payload.putFloat(roll_rate);
53:         packet.payload.putFloat(pitch_rate);
54:         packet.payload.putFloat(yaw_rate);
55:         packet.payload.putFloat(thrust);
56:         return packet;
57:
58:     /**
59:     * Decode a roll_pitch_yaw_rates_thrust_setpoint message into this class field
S
60:     *
61:     * @param payload The message to decode
62:     */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         time_boot_ms = payload.getInt();
66:         roll_rate = payload.getFloat();
67:         pitch_rate = payload.getFloat();
68:         yaw_rate = payload.getFloat();
69:         thrust = payload.getFloat();
70:     }
71:
72:     /**
73:     * Constructor for a new message, just initializes the msgid
74:     */
75:     public msg_roll_pitch_yaw_rates_thrust_setpoint(){
76:         msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT;
77:     }
78:
79:     /**
80:     * Constructor for a new message, initializes the message with the payload
81:     * from a mavlink packet
82:     */
83:     /**
84:     public msg_roll_pitch_yaw_rates_thrust_setpoint(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compuid = mavLinkPacket.compuid;
87:         this.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "ROLL_PITCH_YAW_RATES_THRUST_SETPOINT");
90:         //Log.d("MAVLINK_MSG_ID_ROLL_PITCH_YAW_RATES_THRUST_SETPOINT", toString())
;
91:     }
92:
93:
94:     /**
95:     * Returns a string with the MSG name and data
96:     */
97:     public String toString(){
98:         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_rat
e+" thrust:"+thrust+"";
99:     }
100: }

```



```

1: // MESSAGE SET_MODE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set the system mode, as defined by enum MAV_MODE. There is no target component i
11:  * as the mode is by definition for the overall aircraft, not only for one component.
12:  */
13: public class msg_set_mode extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_SET_MODE = 11;
16:     public static final int MAVLINK_MSG_LENGTH = 6;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_MODE;
18:
19:     /**
20:      * The new autopilot-specific mode. This field can be ignored by an autopil
21:      */
22:     public int custom_mode;
23:     /**
24:      * The system setting the mode
25:      */
26:     public byte target_system;
27:     /**
28:      * The new base mode
29:      */
30:     public byte base_mode;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compuid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_SET_MODE;
42:         packet.payload.putInt(custom_mode);
43:         packet.payload.putByte(target_system);
44:         packet.payload.putByte(base_mode);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a set_mode message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         custom_mode = payload.getInt();
56:         target_system = payload.getByte();
57:         base_mode = payload.getByte();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_set_mode(){
64:         msgid = MAVLINK_MSG_ID_SET_MODE;
65:     }
66:
67:     /**
68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      *
71:      */
72:     public msg_set_mode(MAVLinkPacket mavLinkPacket){
73:         this.sysid = mavLinkPacket.sysid;
74:         this.compuid = mavLinkPacket.compuid;
75:         this.msgid = MAVLINK_MSG_ID_SET_MODE;
76:         unpack(mavLinkPacket.payload);
77:         //Log.d("MAVLink", "SET_MODE");
78:         //Log.d("MAVLINK_MSG_ID_SET_MODE", toString());
79:     }
80:
81:     /**
82:      * Returns a string with the MSG name and data
83:      */
84:     public String toString(){
85:         return "MAVLINK_MSG_ID_SET_MODE -"+" custom_mode:"+custom_mode+" target_sy
86: stem:"+target_system+" base_mode:"+base_mode+"";
87:     }
88: }

```



```

1: // MESSAGE NAV_CONTROLLER_OUTPUT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Outputs of the APM navigation controller. The primary use of this message is to
11:  * check the response and signs of the controller before actual flight and to assist with tuning
12:  * controller parameters.
13:  */
14: public class msg_nav_controller_output extends MAVLinkMessage{
15:
16:     public static final int MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT = 62;
17:     public static final int MAVLINK_MSG_LENGTH = 26;
18:     private static final long serialVersionUID = MAVLINK_MSG_ID_NAV_CONTROLLER
19:     _OUTPUT;
20:
21:     /**
22:      * Current desired roll in degrees
23:      */
24:     public float nav_roll;
25:
26:     /**
27:      * Current desired pitch in degrees
28:      */
29:     public float nav_pitch;
30:
31:     /**
32:      * Current altitude error in meters
33:      */
34:     public float alt_error;
35:
36:     /**
37:      * Current airspeed error in meters/second
38:      */
39:     public float aspd_error;
40:
41:     /**
42:      * Current crosstrack error on x-y plane in meters
43:      */
44:     public float xtrack_error;
45:
46:     /**
47:      * Current desired heading in degrees
48:      */
49:     public short nav_bearing;
50:
51:     /**
52:      * Bearing to current MISSION/target in degrees
53:      */
54:     public short target_bearing;
55:
56:     /**
57:      * Distance to active MISSION in meters
58:      */
59:     public short wp_dist;
60:
61:     /**
62:      * Generates the payload for a mavlink message for a message of this type
63:      * @return
64:      */
65:     public MAVLinkPacket pack(){
66:         MAVLinkPacket packet = new MAVLinkPacket();
67:         packet.len = MAVLINK_MSG_LENGTH;
68:         packet.sysid = 255;
69:         packet.compid = 190;
70:         packet.msgid = MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT;
71:         packet.payload.putFloat(nav_roll);
72:         packet.payload.putFloat(nav_pitch);
73:         packet.payload.putFloat(alt_error);

```

```

65:         packet.payload.putFloat(aspd_error);
66:         packet.payload.putFloat(xtrack_error);
67:         packet.payload.putShort(nav_bearing);
68:         packet.payload.putShort(target_bearing);
69:         packet.payload.putShort(wp_dist);
70:         return packet;
71:     }
72:
73:     /**
74:      * Decode a nav_controller_output message into this class fields
75:      */
76:     * @param payload The message to decode
77:     */
78:     public void unpack(MAVLinkPayload payload) {
79:         payload.resetIndex();
80:         nav_roll = payload.getFloat();
81:         nav_pitch = payload.getFloat();
82:         alt_error = payload.getFloat();
83:         aspd_error = payload.getFloat();
84:         xtrack_error = payload.getFloat();
85:         nav_bearing = payload.getShort();
86:         target_bearing = payload.getShort();
87:         wp_dist = payload.getShort();
88:     }
89:
90:     /**
91:      * Constructor for a new message, just initializes the msgid
92:      */
93:     public msg_nav_controller_output(){
94:         msgid = MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT;
95:     }
96:
97:     /**
98:      * Constructor for a new message, initializes the message with the payload
99:      * from a mavlink packet
100:      */
101:     public msg_nav_controller_output(MAVLinkPacket mavLinkPacket){
102:         this.sysid = mavLinkPacket.sysid;
103:         this.compid = mavLinkPacket.compid;
104:         this.msgid = MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT;
105:         unpack(mavLinkPacket.payload);
106:         //Log.d("MAVLink", "NAV_CONTROLLER_OUTPUT");
107:         //Log.d("MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT", toString());
108:     }
109:
110:     /**
111:      * Returns a string with the MSG name and data
112:      */
113:     public String toString(){
114:         return "MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPUT -"+" nav_roll:"+nav_roll+" na
115:         v_pitch:"+nav_pitch+" alt_error:"+alt_error+" aspd_error:"+aspd_error+" xtrack_error:"+xt
116:         rack_error+" nav_bearing:"+nav_bearing+" target_bearing:"+target_bearing+" wp_dist:"+wp_d
117:         ist+"";
118:     }

```



```

1: // MESSAGE OPTICAL_FLOW PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * Optical flow from a flow sensor (e.g. optical mouse sensor)
11: */
12: public class msg_optical_flow extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_OPTICAL_FLOW = 100;
15:     public static final int MAVLINK_MSG_LENGTH = 26;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_OPTICAL_FLOW;
17:
18:
19:     /**
20:     * Timestamp (UNIX)
21:     */
22:     public long time_usec;
23:     /**
24:     * Flow in meters in x-sensor direction, angular-speed compensated
25:     */
26:     public float flow_comp_m_x;
27:     /**
28:     * Flow in meters in y-sensor direction, angular-speed compensated
29:     */
30:     public float flow_comp_m_y;
31:     /**
32:     * Ground distance in meters. Positive value: distance known. Negative value: Unknown distance
33:     */
34:     public float ground_distance;
35:     /**
36:     * Flow in pixels in x-sensor direction
37:     */
38:     public short flow_x;
39:     /**
40:     * Flow in pixels in y-sensor direction
41:     */
42:     public short flow_y;
43:     /**
44:     * Sensor ID
45:     */
46:     public byte sensor_id;
47:     /**
48:     * Optical flow quality / confidence. 0: bad, 255: maximum quality
49:     */
50:     public byte quality;
51:
52:     /**
53:     * Generates the payload for a mavlink message for a message of this type
54:     * @return
55:     */
56:     public MAVLinkPacket pack(){
57:         MAVLinkPacket packet = new MAVLinkPacket();
58:         packet.len = MAVLINK_MSG_LENGTH;
59:         packet.sysid = 255;
60:         packet.compид = 190;
61:         packet.msgid = MAVLINK_MSG_ID_OPTICAL_FLOW;
62:         packet.payload.putLong(time_usec);
63:         packet.payload.putFloat(flow_comp_m_x);
64:         packet.payload.putFloat(flow_comp_m_y);
65:         packet.payload.putFloat(ground_distance);
66:         packet.payload.putShort(flow_x);

```

```

67:         packet.payload.putShort(flow_y);
68:         packet.payload.putByte(sensor_id);
69:         packet.payload.putByte(quality);
70:         return packet;
71:     }
72:
73:     /**
74:     * Decode a optical_flow message into this class fields
75:     */
76:     * @param payload The message to decode
77:     */
78:     public void unpack(MAVLinkPayload payload) {
79:         payload.resetIndex();
80:         time_usec = payload.getLong();
81:         flow_comp_m_x = payload.getFloat();
82:         flow_comp_m_y = payload.getFloat();
83:         ground_distance = payload.getFloat();
84:         flow_x = payload.getShort();
85:         flow_y = payload.getShort();
86:         sensor_id = payload.getByte();
87:         quality = payload.getByte();
88:     }
89:
90:     /**
91:     * Constructor for a new message, just initializes the msgid
92:     */
93:     public msg_optical_flow(){
94:         msgid = MAVLINK_MSG_ID_OPTICAL_FLOW;
95:     }
96:
97:     /**
98:     * Constructor for a new message, initializes the message with the payload
99:     * from a mavlink packet
100:     */
101:
102:     public msg_optical_flow(MAVLinkPacket mavLinkPacket){
103:         this.sysid = mavLinkPacket.sysid;
104:         this.compид = mavLinkPacket.compид;
105:         this.msgid = MAVLINK_MSG_ID_OPTICAL_FLOW;
106:         unpack(mavLinkPacket.payload);
107:         //Log.d("MAVLink", "OPTICAL_FLOW");
108:         //Log.d("MAVLINK_MSG_ID_OPTICAL_FLOW", toString());
109:     }
110:
111:
112:     /**
113:     * Returns a string with the MSG name and data
114:     */
115:     public String toString(){
116:         return "MAVLINK_MSG_ID_OPTICAL_FLOW -" + time_usec + " 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 + "";
117:     }
118: }

```


./com/MAVLink/Messages/ardupilotmega/msg_param_request_list.java

```
1: // MESSAGE PARAM_REQUEST_LIST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Request all parameters of this component. After his request, all parameters are
emitted.
11:  */
12: public class msg_param_request_list extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_PARAM_REQUEST_LIST = 21;
15:     public static final int MAVLINK_MSG_LENGTH = 2;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_REQUEST_
LIST;
17:
18:
19:     /**
20:      * System ID
21:      */
22:     public byte target_system;
23:     /**
24:      * Component ID
25:      */
26:     public byte target_component;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compид = 190;
37:         packet.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_LIST;
38:         packet.payload.putByte(target_system);
39:         packet.payload.putByte(target_component);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a param_request_list message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         target_system = payload.getByte();
51:         target_component = payload.getByte();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_param_request_list(){
58:         msgid = MAVLINK_MSG_ID_PARAM_REQUEST_LIST;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      *
65:      */
```

Fri Oct 25 14:10:51 2013

1

```
66:     public msg_param_request_list(MAVLinkPacket mavLinkPacket){
67:         this.sysid = mavLinkPacket.sysid;
68:         this.compид = mavLinkPacket.compид;
69:         this.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_LIST;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "PARAM_REQUEST_LIST");
72:         //Log.d("MAVLINK_MSG_ID_PARAM_REQUEST_LIST", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_PARAM_REQUEST_LIST -" + target_system:" +target_syst
em+" target_component:" +target_component+"";
81:     }
82: }
```



```

1: // MESSAGE PARAM_REQUEST_READ PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Request to read the onboard parameter with the param_id string id. Onboard param
eters 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 autopilot
s. See also http://qgroundcontrol.org/parameter\_interface for a full documentation of QGr
oundControl and IMU code.
11: */
12: public class msg_param_request_read extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_PARAM_REQUEST_READ = 20;
15:     public static final int MAVLINK_MSG_LENGTH = 20;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_REQUEST_
READ;
17:
18:
19:     /**
20:      * Parameter index. Send -1 to use the param ID field as identifier (else t
he param id will be ignored)
21:      */
22:     public short param_index;
23:     /**
24:      * System ID
25:      */
26:     public byte target_system;
27:     /**
28:      * Component ID
29:      */
30:     public byte target_component;
31:     /**
32:      * Onboard parameter id, terminated by NULL if the length is less than 16 h
uman-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
33:      */
34:     public byte param_id[] = new byte[16];
35:
36:     /**
37:      * Generates the payload for a mavlink message for a message of this type
38:      * @return
39:      */
40:     public MAVLinkPacket pack(){
41:         MAVLinkPacket packet = new MAVLinkPacket();
42:         packet.len = MAVLINK_MSG_LENGTH;
43:         packet.sysid = 255;
44:         packet.compid = 190;
45:         packet.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_READ;
46:         packet.payload.putShort(param_index);
47:         packet.payload.putByte(target_system);
48:         packet.payload.putByte(target_component);
49:         for (int i = 0; i < param_id.length; i++) {
50:             packet.payload.putByte(param_id[i]);
51:         }
52:         return packet;
53:     }
54:
55:     /**
56:      * Decode a param_request_read message into this class fields
57:      *
58:      * @param payload The message to decode

```

```

59:     */
60:     public void unpack(MAVLinkPayload payload) {
61:         payload.resetIndex();
62:         param_index = payload.getShort();
63:         target_system = payload.getByte();
64:         target_component = payload.getByte();
65:         for (int i = 0; i < param_id.length; i++) {
66:             param_id[i] = payload.getByte();
67:         }
68:     }
69:
70:     /**
71:      * Constructor for a new message, just initializes the msgid
72:      */
73:     public msg_param_request_read(){
74:         msgid = MAVLINK_MSG_ID_PARAM_REQUEST_READ;
75:     }
76:
77:     /**
78:      * Constructor for a new message, initializes the message with the payload
79:      * from a mavlink packet
80:      */
81:     /**
82:     public msg_param_request_read(MAVLinkPacket mavLinkPacket){
83:         this.sysid = mavLinkPacket.sysid;
84:         this.compid = mavLinkPacket.compid;
85:         this.msgid = MAVLINK_MSG_ID_PARAM_REQUEST_READ;
86:         unpack(mavLinkPacket.payload);
87:         //Log.d("MAVLink", "PARAM_REQUEST_READ");
88:         //Log.d("MAVLINK_MSG_ID_PARAM_REQUEST_READ", toString());
89:     }
90:
91:     /**
92:      * Sets the buffer of this message with a string, adds the necessary padding
93:      */
94:     public void setParam_Id(String str) {
95:         int len = Math.min(str.length(), 16);
96:         for (int i=0; i<len; i++) {
97:             param_id[i] = (byte) str.charAt(i);
98:         }
99:         for (int i=len; i<16; i++) {
100:             param_id[i] = 0;
101:         }
102:     }
103:
104:     /**
105:      * Gets the message, formatted as a string
106:      */
107:     public String getParam_Id() {
108:         String result = "";
109:         for (int i = 0; i < 16; i++) {
110:             if (param_id[i] != 0)
111:                 result = result + (char) param_id[i];
112:             else
113:                 break;
114:         }
115:         return result;
116:     }
117:
118:     /**
119:      * Returns a string with the MSG name and data
120:      */
121:     public String toString(){
122:         return "MAVLINK_MSG_ID_PARAM_REQUEST_READ -"+" param_index:"+param_index+"
target_system:"+target_system+" target_component:"+target_component+" param_id:"+param_i
d+"";

```

./com/MAVLink/Messages/ardupilotmega/msg_param_request_read.java

Fri Oct 25 14:10:51 2013

2

```
123:     }  
124: }
```

```

1: // MESSAGE PARAM_SET PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set a parameter value TEMPORARILY to RAM. It will be reset to default on system
11:  * reboot. Send the ACTION MAV_ACTION_STORAGE_WRITE to PERMANENTLY write the RAM contents to
12:  * EEPROM. IMPORTANT: The receiving component should acknowledge the new parameter value by
13:  * sending a param_value message to all communication partners. This will also ensure that
14:  * multiple GCS all have an up-to-date list of all parameters. If the sending GCS did not re
15:  * ceive a PARAM_VALUE message within its timeout time, it should re-send the PARAM_SET mess
16:  * age.
17:  */
18: public class msg_param_set extends MAVLinkMessage{
19:
20:     public static final int MAVLINK_MSG_ID_PARAM_SET = 23;
21:     public static final int MAVLINK_MSG_LENGTH = 23;
22:     private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_SET;
23:
24:     /**
25:      * Onboard parameter value
26:      */
27:     public float param_value;
28:     /**
29:      * System ID
30:      */
31:     public byte target_system;
32:     /**
33:      * Component ID
34:      */
35:     public byte target_component;
36:     /**
37:      * Onboard parameter id, terminated by NULL if the length is less than 16 h
38:      * uman-readable chars and WITHOUT null termination (NULL) byte if the length is exactly 16
39:      * chars - applications have to provide 16+1 bytes storage if the ID is stored as string
40:      */
41:     public byte param_id[] = new byte[16];
42:     /**
43:      * Onboard parameter type: see the MAV_PARAM_TYPE enum for supported data t
44:      * ypes.
45:      */
46:     public byte param_type;
47:
48:     /**
49:      * Generates the payload for a mavlink message for a message of this type
50:      * @return
51:      */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_PARAM_SET;
58:         packet.payload.putFloat(param_value);
59:         packet.payload.putByte(target_system);
60:         packet.payload.putByte(target_component);
61:         for (int i = 0; i < param_id.length; i++) {
62:             packet.payload.putByte(param_id[i]);
63:         }
64:         packet.payload.putByte(param_type);
65:         return packet;
66:     }

```

```

59:
60: /**
61:  * Decode a param_set message into this class fields
62:  *
63:  * @param payload The message to decode
64:  */
65: public void unpack(MAVLinkPayload payload) {
66:     payload.resetIndex();
67:     param_value = payload.getFloat();
68:     target_system = payload.getByte();
69:     target_component = payload.getByte();
70:     for (int i = 0; i < param_id.length; i++) {
71:         param_id[i] = payload.getByte();
72:     }
73:     param_type = payload.getByte();
74: }
75:
76: /**
77:  * Constructor for a new message, just initializes the msgid
78:  */
79: public msg_param_set(){
80:     msgid = MAVLINK_MSG_ID_PARAM_SET;
81: }
82:
83: /**
84:  * Constructor for a new message, initializes the message with the payload
85:  * from a mavlink packet
86:  */
87: public msg_param_set(MAVLinkPacket mavLinkPacket){
88:     this.sysid = mavLinkPacket.sysid;
89:     this.compid = mavLinkPacket.compid;
90:     this.msgid = MAVLINK_MSG_ID_PARAM_SET;
91:     unpack(mavLinkPacket.payload);
92:     //Log.d("MAVLink", "PARAM_SET");
93:     //Log.d("MAVLINK_MSG_ID_PARAM_SET", toString());
94: }
95:
96: /**
97:  * Sets the buffer of this message with a string, adds the necessary padding
98:  */
99: public void setParam_Id(String str) {
100:     int len = Math.min(str.length(), 16);
101:     for (int i=0; i<len; i++) {
102:         param_id[i] = (byte) str.charAt(i);
103:     }
104:     for (int i=len; i<16; i++) {
105:         // padding for the rest of
106:         param_id[i] = 0;
107:     }
108: }
109:
110: /**
111:  * Gets the message, formatted as a string
112:  */
113: public String getParam_Id() {
114:     String result = "";
115:     for (int i = 0; i < 16; i++) {
116:         if (param_id[i] != 0)
117:             result = result + (char) param_id[i];
118:         else
119:             break;
120:     }
121:     return result;
122: }
123:
124: /**

```

```
125:      * Returns a string with the MSG name and data
126:      */
127:      public String toString(){
128:          return "MAVLINK_MSG_ID_PARAM_SET -"+" param_value:"+param_value+" target_s
system:"+target_system+" target_component:"+target_component+" param_id:"+param_id+" param
_type:"+param_type+"";
129:      }
130: }
```

```

1: // MESSAGE PARAM_VALUE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Emit the value of a onboard parameter. The inclusion of param_count and param_in
dex in the message allows the recipient to keep track of received parameters and allows h
im to re-request missing parameters after a loss or timeout.
11: */
12: public class msg_param_value extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_PARAM_VALUE = 22;
15:     public static final int MAVLINK_MSG_LENGTH = 25;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_PARAM_VALUE;
17:
18:     /**
19:     * Onboard parameter value
20:     */
21:     public float param_value;
22:     /**
23:     * Total number of onboard parameters
24:     */
25:     public short param_count;
26:     /**
27:     * Index of this onboard parameter
28:     */
29:     public short param_index;
30:     /**
31:     * Onboard parameter id, terminated by NULL if the length is less than 16 h
uman-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
32:     */
33:     public byte param_id[] = new byte[16];
34:     /**
35:     * Onboard parameter type: see the MAV_PARAM_TYPE enum for supported data t
ypes.
36:     */
37:     public byte param_type;
38:
39:     /**
40:     * Generates the payload for a mavlink message for a message of this type
41:     * @return
42:     */
43:     public MAVLinkPacket pack(){
44:         MAVLinkPacket packet = new MAVLinkPacket();
45:         packet.len = MAVLINK_MSG_LENGTH;
46:         packet.sysid = 255;
47:         packet.compid = 190;
48:         packet.msgid = MAVLINK_MSG_ID_PARAM_VALUE;
49:         packet.payload.putFloat(param_value);
50:         packet.payload.putShort(param_count);
51:         packet.payload.putShort(param_index);
52:         for (int i = 0; i < param_id.length; i++) {
53:             packet.payload.putByte(param_id[i]);
54:         }
55:         packet.payload.putByte(param_type);
56:         return packet;
57:     }
58:
59:     /**
60:     * Decode a param_value message into this class fields
61:     */
62:

```

```

63:     * @param payload The message to decode
64:     */
65:     public void unpack(MAVLinkPayload payload) {
66:         payload.resetIndex();
67:         param_value = payload.getFloat();
68:         param_count = payload.getShort();
69:         param_index = payload.getShort();
70:         for (int i = 0; i < param_id.length; i++) {
71:             param_id[i] = payload.getByte();
72:         }
73:         param_type = payload.getByte();
74:     }
75:
76:     /**
77:     * Constructor for a new message, just initializes the msgid
78:     */
79:     public msg_param_value(){
80:         msgid = MAVLINK_MSG_ID_PARAM_VALUE;
81:     }
82:
83:     /**
84:     * Constructor for a new message, initializes the message with the payload
85:     * from a mavlink packet
86:     */
87:     public msg_param_value(MAVLinkPacket mavLinkPacket){
88:         this.sysid = mavLinkPacket.sysid;
89:         this.compid = mavLinkPacket.compid;
90:         this.msgid = MAVLINK_MSG_ID_PARAM_VALUE;
91:         unpack(mavLinkPacket.payload);
92:         //Log.d("MAVLink", "PARAM_VALUE");
93:         //Log.d("MAVLINK_MSG_ID_PARAM_VALUE", toString());
94:     }
95:
96:     /**
97:     * Sets the buffer of this message with a string, adds the necessary padding
98:     */
99:     public void setParam_Id(String str) {
100:         int len = Math.min(str.length(), 16);
101:         for (int i=0; i<len; i++) {
102:             param_id[i] = (byte) str.charAt(i);
103:         }
104:         for (int i=len; i<16; i++) {
105:             // padding for the rest of
the buffer
106:             param_id[i] = 0;
107:         }
108:     }
109:
110:     /**
111:     * Gets the message, formatted as a string
112:     */
113:     public String getParam_Id() {
114:         String result = "";
115:         for (int i = 0; i < 16; i++) {
116:             if (param_id[i] != 0)
117:                 result = result + (char) param_id[i];
118:             else
119:                 break;
120:         }
121:         return result;
122:     }
123:
124:     /**
125:     * Returns a string with the MSG name and data
126:     */
127:     public String toString(){
128:         return "MAVLINK_MSG_ID_PARAM_VALUE -"+" param_value:"+param_value+" param_

```

```
count:"+param_count+" param_index:"+param_index+" param_id:"+param_id+" param_type:"+param_type+"";
129:      }
130: }
```

```

1: // MESSAGE PING PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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.
11:  */
12: public class msg_ping extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_PING = 4;
15:     public static final int MAVLINK_MSG_LENGTH = 14;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_PING;
17:
18:
19:     /**
20:      * Unix timestamp in microseconds
21:      */
22:     public long time_usec;
23:     /**
24:      * PING sequence
25:      */
26:     public int seq;
27:     /**
28:      * 0: request ping from all receiving systems, if greater than 0: message i
s a ping response and number is the system id of the requesting system
29:      */
30:     public byte target_system;
31:     /**
32:      * 0: request ping from all receiving components, if greater than 0: messag
e is a ping response and number is the system id of the requesting system
33:      */
34:     public byte target_component;
35:
36:     /**
37:      * Generates the payload for a mavlink message for a message of this type
38:      * @return
39:      */
40:     public MAVLinkPacket pack(){
41:         MAVLinkPacket packet = new MAVLinkPacket();
42:         packet.len = MAVLINK_MSG_LENGTH;
43:         packet.sysid = 255;
44:         packet.compid = 190;
45:         packet.msgid = MAVLINK_MSG_ID_PING;
46:         packet.payload.putLong(time_usec);
47:         packet.payload.putInt(seq);
48:         packet.payload.putByte(target_system);
49:         packet.payload.putByte(target_component);
50:         return packet;
51:     }
52:
53:     /**
54:      * Decode a ping message into this class fields
55:      *
56:      * @param payload The message to decode
57:      */
58:     public void unpack(MAVLinkPayload payload) {
59:         payload.resetIndex();
60:         time_usec = payload.getLong();
61:         seq = payload.getInt();
62:         target_system = payload.getByte();
63:         target_component = payload.getByte();
64:     }
65:
66:     /**
67:      * Constructor for a new message, just initializes the msgid
68:      */
69:     public msg_ping(){
70:         msgid = MAVLINK_MSG_ID_PING;
71:     }
72:
73:     /**
74:      * Constructor for a new message, initializes the message with the payload
75:      * from a mavlink packet
76:      */
77:     /**
78:      * public msg_ping(MAVLinkPacket mavLinkPacket){
79:      *     this.sysid = mavLinkPacket.sysid;
80:      *     this.compid = mavLinkPacket.compid;
81:      *     this.msgid = MAVLINK_MSG_ID_PING;
82:      *     unpack(mavLinkPacket.payload);
83:      *     //Log.d("MAVLink", "PING");
84:      *     //Log.d("MAVLINK_MSG_ID_PING", toString());
85:      * }
86:
87:
88:     /**
89:      * Returns a string with the MSG name and data
90:      */
91:     public String toString(){
92:         return "MAVLINK_MSG_ID_PING -"+" time_usec:"+time_usec+" seq:"+seq+" targe
t_system:"+target_system+" target_component:"+target_component+"";
93:     }
94: }

```



```

1: // MESSAGE RADIO PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Status generated by radio
11:  */
12: public class msg_radio extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_RADIO = 166;
15:     public static final int MAVLINK_MSG_LENGTH = 9;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_RADIO;
17:
18:
19:     /**
20:     * receive errors
21:     */
22:     public short rxerrors;
23:     /**
24:     * count of error corrected packets
25:     */
26:     public short fixed;
27:     /**
28:     * local signal strength
29:     */
30:     public byte rssi;
31:     /**
32:     * remote signal strength
33:     */
34:     public byte remrssi;
35:     /**
36:     * how full the tx buffer is as a percentage
37:     */
38:     public byte txbuf;
39:     /**
40:     * background noise level
41:     */
42:     public byte noise;
43:     /**
44:     * remote background noise level
45:     */
46:     public byte remnoise;
47:
48:     /**
49:     * Generates the payload for a mavlink message for a message of this type
50:     * @return
51:     */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compuid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_RADIO;
58:         packet.payload.putShort(rxerrors);
59:         packet.payload.putShort(fixed);
60:         packet.payload.putByte(rssi);
61:         packet.payload.putByte(remrssi);
62:         packet.payload.putByte(txbuf);
63:         packet.payload.putByte(noise);
64:         packet.payload.putByte(remnoise);
65:         return packet;
66:     }
67:

```

```

68:     /**
69:     * Decode a radio message into this class fields
70:     *
71:     * @param payload The message to decode
72:     */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         rxerrors = payload.getShort();
76:         fixed = payload.getShort();
77:         rssi = payload.getByte();
78:         remrssi = payload.getByte();
79:         txbuf = payload.getByte();
80:         noise = payload.getByte();
81:         remnoise = payload.getByte();
82:     }
83:
84:     /**
85:     * Constructor for a new message, just initializes the msgid
86:     */
87:     public msg_radio(){
88:         msgid = MAVLINK_MSG_ID_RADIO;
89:     }
90:
91:     /**
92:     * Constructor for a new message, initializes the message with the payload
93:     * from a mavlink packet
94:     *
95:     */
96:     public msg_radio(MAVLinkPacket mavLinkPacket){
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compuid = mavLinkPacket.compuid;
99:         this.msgid = MAVLINK_MSG_ID_RADIO;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "RADIO");
102:         //Log.d("MAVLINK_MSG_ID_RADIO", toString());
103:     }
104:
105:
106:     /**
107:     * Returns a string with the MSG name and data
108:     */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_RADIO -"+" rxerrors:"+rxerrors+" fixed:"+fixed+" rs
si:"+rssi+" remrssi:"+remrssi+" txbuf:"+txbuf+" noise:"+noise+" remnoise:"+remnoise+"";
111:     }
112: }

```



```

1: // MESSAGE_RAW_IMU_PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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 debuggi
    ng.
11: */
12: public class msg_raw_imu extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_RAW_IMU = 27;
15:     public static final int MAVLINK_MSG_LENGTH = 26;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_RAW_IMU;
17:
18:
19:     /**
20:      * Timestamp (microseconds since UNIX epoch or microseconds since system bo
    ot)
21:      */
22:     public long time_usec;
23:     /**
24:      * X acceleration (raw)
25:      */
26:     public short xacc;
27:     /**
28:      * Y acceleration (raw)
29:      */
30:     public short yacc;
31:     /**
32:      * Z acceleration (raw)
33:      */
34:     public short zacc;
35:     /**
36:      * Angular speed around X axis (raw)
37:      */
38:     public short xgyro;
39:     /**
40:      * Angular speed around Y axis (raw)
41:      */
42:     public short ygyro;
43:     /**
44:      * Angular speed around Z axis (raw)
45:      */
46:     public short zgyro;
47:     /**
48:      * X Magnetic field (raw)
49:      */
50:     public short xmag;
51:     /**
52:      * Y Magnetic field (raw)
53:      */
54:     public short ymag;
55:     /**
56:      * Z Magnetic field (raw)
57:      */
58:     public short zmag;
59:
60:     /**
61:      * Generates the payload for a mavlink message for a message of this type
62:      * @return
63:      */
64:     public MAVLinkPacket pack(){

```

```

65:         MAVLinkPacket packet = new MAVLinkPacket();
66:         packet.len = MAVLINK_MSG_LENGTH;
67:         packet.sysid = 255;
68:         packet.compid = 190;
69:         packet.msgid = MAVLINK_MSG_ID_RAW_IMU;
70:         packet.payload.putLong(time_usec);
71:         packet.payload.putShort(xacc);
72:         packet.payload.putShort(yacc);
73:         packet.payload.putShort(zacc);
74:         packet.payload.putShort(xgyro);
75:         packet.payload.putShort(ygyro);
76:         packet.payload.putShort(zgyro);
77:         packet.payload.putShort(xmag);
78:         packet.payload.putShort(ymag);
79:         packet.payload.putShort(zmag);
80:         return packet;
81:     }
82:
83:     /**
84:      * Decode a raw_imu message into this class fields
85:      *
86:      * @param payload The message to decode
87:      */
88:     public void unpack(MAVLinkPayload payload) {
89:         payload.resetIndex();
90:         time_usec = payload.getLong();
91:         xacc = payload.getShort();
92:         yacc = payload.getShort();
93:         zacc = payload.getShort();
94:         xgyro = payload.getShort();
95:         ygyro = payload.getShort();
96:         zgyro = payload.getShort();
97:         xmag = payload.getShort();
98:         ymag = payload.getShort();
99:         zmag = payload.getShort();
100:     }
101:
102:     /**
103:      * Constructor for a new message, just initializes the msgid
104:      */
105:     public msg_raw_imu(){
106:         msgid = MAVLINK_MSG_ID_RAW_IMU;
107:     }
108:
109:     /**
110:      * Constructor for a new message, initializes the message with the payload
111:      * from a mavlink packet
112:      */
113:     public msg_raw_imu(MAVLinkPacket mavLinkPacket){
114:         this.sysid = mavLinkPacket.sysid;
115:         this.compid = mavLinkPacket.compid;
116:         this.msgid = MAVLINK_MSG_ID_RAW_IMU;
117:         unpack(mavLinkPacket.payload);
118:         //Log.d("MAVLink", "RAW_IMU");
119:         //Log.d("MAVLINK_MSG_ID_RAW_IMU", toString());
120:     }
121:
122:
123:
124:     /**
125:      * Returns a string with the MSG name and data
126:      */
127:     public String toString(){
128:         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+"";
129:     }

```

./com/MAVLink/Messages/ardupilotmega/msg_raw_imu.java

Fri Oct 25 14:10:51 2013

2

130: }

./com/MAVLink/Messages/ardupilotmega/msg_raw_pressure.java

```
1: // MESSAGE RAW_PRESSURE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The RAW pressure readings for the typical setup of one absolute pressure and one
11: * differential pressure sensor. The sensor values should be the raw, UNSCALED ADC values.
12: */
13: public class msg_raw_pressure extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_RAW_PRESSURE = 28;
16:     public static final int MAVLINK_MSG_LENGTH = 16;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_RAW_PRESSURE;
18:
19:     /**
20:     * Timestamp (microseconds since UNIX epoch or microseconds since system bo
21:     */
22:     public long time_usec;
23:     /**
24:     * Absolute pressure (raw)
25:     */
26:     public short press_abs;
27:     /**
28:     * Differential pressure 1 (raw)
29:     */
30:     public short press_diff1;
31:     /**
32:     * Differential pressure 2 (raw)
33:     */
34:     public short press_diff2;
35:     /**
36:     * Raw Temperature measurement (raw)
37:     */
38:     public short temperature;
39:
40:     /**
41:     * Generates the payload for a mavlink message for a message of this type
42:     * @return
43:     */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_RAW_PRESSURE;
50:         packet.payload.putLong(time_usec);
51:         packet.payload.putShort(press_abs);
52:         packet.payload.putShort(press_diff1);
53:         packet.payload.putShort(press_diff2);
54:         packet.payload.putShort(temperature);
55:         return packet;
56:     }
57:
58:     /**
59:     * Decode a raw_pressure message into this class fields
60:     *
61:     * @param payload The message to decode
62:     */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         time_usec = payload.getLong();
```

Fri Oct 25 14:10:51 2013

1

```
66:         press_abs = payload.getShort();
67:         press_diff1 = payload.getShort();
68:         press_diff2 = payload.getShort();
69:         temperature = payload.getShort();
70:     }
71:
72:     /**
73:     * Constructor for a new message, just initializes the msgid
74:     */
75:     public msg_raw_pressure(){
76:         msgid = MAVLINK_MSG_ID_RAW_PRESSURE;
77:     }
78:
79:     /**
80:     * Constructor for a new message, initializes the message with the payload
81:     * from a mavlink packet
82:     */
83:     public msg_raw_pressure(MAVLinkPacket mavLinkPacket){
84:         this.sysid = mavLinkPacket.sysid;
85:         this.compid = mavLinkPacket.compid;
86:         this.msgid = MAVLINK_MSG_ID_RAW_PRESSURE;
87:         unpack(mavLinkPacket.payload);
88:         //Log.d("MAVLink", "RAW_PRESSURE");
89:         //Log.d("MAVLINK_MSG_ID_RAW_PRESSURE", toString());
90:     }
91:
92:
93:     /**
94:     * Returns a string with the MSG name and data
95:     */
96:     public String toString(){
97:         return "MAVLINK_MSG_ID_RAW_PRESSURE -"+" time_usec:"+time_usec+" press_abs
98: :"+press_abs+" press_diff1:"+press_diff1+" press_diff2:"+press_diff2+" temperature:"+temp
99: erature+"";
100: }
```



```

1: // MESSAGE RC_CHANNELS_OVERRIDE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * 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 a
s follows: 1000 microseconds: 0%, 2000 microseconds: 100%. Individual receivers/transmitt
ers might violate this specification.
11: */
12: public class msg_rc_channels_override extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE = 70;
15:     public static final int MAVLINK_MSG_LENGTH = 18;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_RC_CHANNELS_OV
ERRIDE;
17:
18:
19:     /**
20:     * RC channel 1 value, in microseconds
21:     */
22:     public short chan1_raw;
23:     /**
24:     * RC channel 2 value, in microseconds
25:     */
26:     public short chan2_raw;
27:     /**
28:     * RC channel 3 value, in microseconds
29:     */
30:     public short chan3_raw;
31:     /**
32:     * RC channel 4 value, in microseconds
33:     */
34:     public short chan4_raw;
35:     /**
36:     * RC channel 5 value, in microseconds
37:     */
38:     public short chan5_raw;
39:     /**
40:     * RC channel 6 value, in microseconds
41:     */
42:     public short chan6_raw;
43:     /**
44:     * RC channel 7 value, in microseconds
45:     */
46:     public short chan7_raw;
47:     /**
48:     * RC channel 8 value, in microseconds
49:     */
50:     public short chan8_raw;
51:     /**
52:     * System ID
53:     */
54:     public byte target_system;
55:     /**
56:     * Component ID
57:     */
58:     public byte target_component;
59:
60:     /**
61:     * Generates the payload for a mavlink message for a message of this type
62:     * @return

```

```

63:
64:     */
65:     public MAVLinkPacket pack(){
66:         MAVLinkPacket packet = new MAVLinkPacket();
67:         packet.len = MAVLINK_MSG_LENGTH;
68:         packet.sysid = 255;
69:         packet.compид = 190;
70:         packet.msgid = MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE;
71:         packet.payload.putShort(chan1_raw);
72:         packet.payload.putShort(chan2_raw);
73:         packet.payload.putShort(chan3_raw);
74:         packet.payload.putShort(chan4_raw);
75:         packet.payload.putShort(chan5_raw);
76:         packet.payload.putShort(chan6_raw);
77:         packet.payload.putShort(chan7_raw);
78:         packet.payload.putShort(chan8_raw);
79:         packet.payload.putByte(target_system);
80:         packet.payload.putByte(target_component);
81:         return packet;
82:     }
83:
84:     /**
85:     * Decode a rc_channels_override message into this class fields
86:     * @param payload The message to decode
87:     */
88:     public void unpack(MAVLinkPayload payload) {
89:         payload.resetIndex();
90:         chan1_raw = payload.getShort();
91:         chan2_raw = payload.getShort();
92:         chan3_raw = payload.getShort();
93:         chan4_raw = payload.getShort();
94:         chan5_raw = payload.getShort();
95:         chan6_raw = payload.getShort();
96:         chan7_raw = payload.getShort();
97:         chan8_raw = payload.getShort();
98:         target_system = payload.getByte();
99:         target_component = payload.getByte();
100:     }
101:
102:     /**
103:     * Constructor for a new message, just initializes the msgid
104:     */
105:     public msg_rc_channels_override(){
106:         msgid = MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE;
107:     }
108:
109:     /**
110:     * Constructor for a new message, initializes the message with the payload
111:     * from a mavlink packet
112:     */
113:     public msg_rc_channels_override(MAVLinkPacket mavLinkPacket){
114:         this.sysid = mavLinkPacket.sysid;
115:         this.compид = mavLinkPacket.compид;
116:         this.msgid = MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE;
117:         unpack(mavLinkPacket.payload);
118:         //Log.d("MAVLink", "RC_CHANNELS_OVERRIDE");
119:         //Log.d("MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE", toString());
120:     }
121:
122:
123:
124:     /**
125:     * Returns a string with the MSG name and data
126:     */
127:     public String toString(){
128:         return "MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE -"+" chan1_raw:"+chan1_raw+" c
han2_raw:"+chan2_raw+" chan3_raw:"+chan3_raw+" chan4_raw:"+chan4_raw+" chan5_raw:"+chan5_

```

./com/MAVLink/Messages/ardupilotmega/msg_rc_channels_override.java

Fri Oct 25 14:10:51 2013

2

```
raw+" chan6_raw:"+chan6_raw+" chan7_raw:"+chan7_raw+" chan8_raw:"+chan8_raw+" target_syst
em:"+target_system+" target_component:"+target_component+"";
129:     }
130: }
```



```

1: // MESSAGE RC_CHANNELS_RAW PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * The RAW values of the RC channels received. The standard PPM modulation is as fo
llows: 1000 microseconds: 0%, 2000 microseconds: 100%. Individual receivers/transmitters
might violate this specification.
11: */
12: public class msg_rc_channels_raw extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_RC_CHANNELS_RAW = 35;
15:     public static final int MAVLINK_MSG_LENGTH = 22;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_RC_CHANNELS_RA
W;
17:
18:
19:     /**
20:      * Timestamp (milliseconds since system boot)
21:      */
22:     public int time_boot_ms;
23:     /**
24:      * RC channel 1 value, in microseconds. A value of 65535 implies the channe
l is unused.
25:      */
26:     public short chan1_raw;
27:     /**
28:      * RC channel 2 value, in microseconds. A value of 65535 implies the channe
l is unused.
29:      */
30:     public short chan2_raw;
31:     /**
32:      * RC channel 3 value, in microseconds. A value of 65535 implies the channe
l is unused.
33:      */
34:     public short chan3_raw;
35:     /**
36:      * RC channel 4 value, in microseconds. A value of 65535 implies the channe
l is unused.
37:      */
38:     public short chan4_raw;
39:     /**
40:      * RC channel 5 value, in microseconds. A value of 65535 implies the channe
l is unused.
41:      */
42:     public short chan5_raw;
43:     /**
44:      * RC channel 6 value, in microseconds. A value of 65535 implies the channe
l is unused.
45:      */
46:     public short chan6_raw;
47:     /**
48:      * RC channel 7 value, in microseconds. A value of 65535 implies the channe
l is unused.
49:      */
50:     public short chan7_raw;
51:     /**
52:      * RC channel 8 value, in microseconds. A value of 65535 implies the channe
l is unused.
53:      */
54:     public short chan8_raw;
55:     /**
56:      * Servo output port (set of 8 outputs = 1 port). Most MAVs will just use o

```

```

ne, but this allows for more than 8 servos.
57:     */
58:     public byte port;
59:     /**
60:      * Receive signal strength indicator, 0: 0%, 100: 100%, 255: invalid/unknow
n.
61:     */
62:     public byte rssi;
63:
64:     /**
65:      * Generates the payload for a mavlink message for a message of this type
66:      * @return
67:      */
68:     public MAVLinkPacket pack(){
69:         MAVLinkPacket packet = new MAVLinkPacket();
70:         packet.len = MAVLINK_MSG_LENGTH;
71:         packet.sysid = 255;
72:         packet.compid = 190;
73:         packet.msgid = MAVLINK_MSG_ID_RC_CHANNELS_RAW;
74:         packet.payload.putInt(time_boot_ms);
75:         packet.payload.putShort(chan1_raw);
76:         packet.payload.putShort(chan2_raw);
77:         packet.payload.putShort(chan3_raw);
78:         packet.payload.putShort(chan4_raw);
79:         packet.payload.putShort(chan5_raw);
80:         packet.payload.putShort(chan6_raw);
81:         packet.payload.putShort(chan7_raw);
82:         packet.payload.putShort(chan8_raw);
83:         packet.payload.putByte(port);
84:         packet.payload.putByte(rssi);
85:         return packet;
86:     }
87:
88:     /**
89:      * Decode a rc_channels_raw message into this class fields
90:      *
91:      * @param payload The message to decode
92:      */
93:     public void unpack(MAVLinkPayload payload) {
94:         payload.resetIndex();
95:         time_boot_ms = payload.getInt();
96:         chan1_raw = payload.getShort();
97:         chan2_raw = payload.getShort();
98:         chan3_raw = payload.getShort();
99:         chan4_raw = payload.getShort();
100:        chan5_raw = payload.getShort();
101:        chan6_raw = payload.getShort();
102:        chan7_raw = payload.getShort();
103:        chan8_raw = payload.getShort();
104:        port = payload.getByte();
105:        rssi = payload.getByte();
106:    }
107:
108:    /**
109:     * Constructor for a new message, just initializes the msgid
110:     */
111:    public msg_rc_channels_raw(){
112:        msgid = MAVLINK_MSG_ID_RC_CHANNELS_RAW;
113:    }
114:
115:    /**
116:     * Constructor for a new message, initializes the message with the payload
117:     * from a mavlink packet
118:     *
119:     */
120:    public msg_rc_channels_raw(MAVLinkPacket mavLinkPacket){
121:        this.sysid = mavLinkPacket.sysid;

```

```

122:         this.compid = mavLinkPacket.compid;
123:         this.msgid = MAVLINK_MSG_ID_RC_CHANNELS_RAW;
124:         unpack(mavLinkPacket.payload);
125:         //Log.d("MAVLink", "RC_CHANNELS_RAW");
126:         //Log.d("MAVLINK_MSG_ID_RC_CHANNELS_RAW", toString());
127:     }
128:
129:
130:     /**
131:      * Returns a string with the MSG name and data
132:      */
133:     public String toString(){
134:         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 + " ";
135:     }
136: }

```

```

1: // MESSAGE RC_CHANNELS_SCALED PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The scaled values of the RC channels received. (-100%) -10000, (0%) 0, (100%) 10000. Channels that are inactive should be set to 65535.
11: */
12: public class msg_rc_channels_scaled extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_RC_CHANNELS_SCALED = 34;
15:     public static final int MAVLINK_MSG_LENGTH = 22;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_RC_CHANNELS_SC
17:
18:     /**
19:     * Timestamp (milliseconds since system boot)
20:     */
21:     public int time_boot_ms;
22:     /**
23:     * RC channel 1 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
24: d) 32767.
25:     */
26:     public short chan1_scaled;
27:     /**
28:     * RC channel 2 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
29: d) 32767.
30:     */
31:     public short chan2_scaled;
32:     /**
33:     * RC channel 3 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
34: d) 32767.
35:     */
36:     public short chan3_scaled;
37:     /**
38:     * RC channel 4 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
39: d) 32767.
40:     */
41:     public short chan4_scaled;
42:     /**
43:     * RC channel 5 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
44: d) 32767.
45:     */
46:     public short chan5_scaled;
47:     /**
48:     * RC channel 6 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
49: d) 32767.
50:     */
51:     public short chan6_scaled;
52:     /**
53:     * RC channel 7 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
54: d) 32767.
55:     */
56:     public short chan7_scaled;
57:     /**
58:     * RC channel 8 value scaled, (-100%) -10000, (0%) 0, (100%) 10000, (invali
59: d) 32767.
60:     */
61:     public short chan8_scaled;
62:     /**
63:     * Servo output port (set of 8 outputs = 1 port). Most MAVs will just use o
64: ne, but this allows for more than 8 servos.

```

```

57:     */
58:     public byte port;
59:     /**
60:     * Receive signal strength indicator, 0: 0%, 100: 100%, 255: invalid/unknow
61: n.
62:     */
63:     public byte rssi;
64:
65:     /**
66:     * Generates the payload for a mavlink message for a message of this type
67:     * @return
68:     */
69:     public MAVLinkPacket pack(){
70:         MAVLinkPacket packet = new MAVLinkPacket();
71:         packet.len = MAVLINK_MSG_LENGTH;
72:         packet.sysid = 255;
73:         packet.compuid = 190;
74:         packet.msgid = MAVLINK_MSG_ID_RC_CHANNELS_SCALED;
75:         packet.payload.putInt(time_boot_ms);
76:         packet.payload.putShort(chan1_scaled);
77:         packet.payload.putShort(chan2_scaled);
78:         packet.payload.putShort(chan3_scaled);
79:         packet.payload.putShort(chan4_scaled);
80:         packet.payload.putShort(chan5_scaled);
81:         packet.payload.putShort(chan6_scaled);
82:         packet.payload.putShort(chan7_scaled);
83:         packet.payload.putShort(chan8_scaled);
84:         packet.payload.putByte(port);
85:         packet.payload.putByte(rssi);
86:         return packet;
87:     }
88:
89:     /**
90:     * Decode a rc_channels_scaled message into this class fields
91:     * @param payload The message to decode
92:     */
93:     public void unpack(MAVLinkPayload payload) {
94:         payload.resetIndex();
95:         time_boot_ms = payload.getInt();
96:         chan1_scaled = payload.getShort();
97:         chan2_scaled = payload.getShort();
98:         chan3_scaled = payload.getShort();
99:         chan4_scaled = payload.getShort();
100:         chan5_scaled = payload.getShort();
101:         chan6_scaled = payload.getShort();
102:         chan7_scaled = payload.getShort();
103:         chan8_scaled = payload.getShort();
104:         port = payload.getByte();
105:         rssi = payload.getByte();
106:     }
107:
108:     /**
109:     * Constructor for a new message, just initializes the msgid
110:     */
111:     public msg_rc_channels_scaled(){
112:         msgid = MAVLINK_MSG_ID_RC_CHANNELS_SCALED;
113:     }
114:
115:     /**
116:     * Constructor for a new message, initializes the message with the payload
117:     * from a mavlink packet
118:     */
119:     public msg_rc_channels_scaled(MAVLinkPacket mavLinkPacket){
120:         this.sysid = mavLinkPacket.sysid;
121:         this.compuid = mavLinkPacket.compuid;

```

```

123:         this.msgid = MAVLINK_MSG_ID_RC_CHANNELS_SCALED;
124:         unpack(mavLinkPacket.payload);
125:         //Log.d("MAVLink", "RC_CHANNELS_SCALED");
126:         //Log.d("MAVLINK_MSG_ID_RC_CHANNELS_SCALED", toString());
127:     }
128:
129:
130:     /**
131:      * Returns a string with the MSG name and data
132:      */
133:     public String toString(){
134:         return "MAVLINK_MSG_ID_RC_CHANNELS_SCALED -"+" time_boot_ms:"+time_boot_ms
135:         +" chan1_scaled:"+chan1_scaled+" chan2_scaled:"+chan2_scaled+" chan3_scaled:"+chan3_scaled+"
136:         chan4_scaled:"+chan4_scaled+" chan5_scaled:"+chan5_scaled+" chan6_scaled:"+chan6_scaled+"
137:         chan7_scaled:"+chan7_scaled+" chan8_scaled:"+chan8_scaled+" port:"+port+" rssi:"+rssi+";
138:     }
139: }

```

```

1:  // MESSAGE REQUEST_DATA_STREAM PACKING
2:  package com.MAVLink.Messages.ardupilotmega;
3:
4:  import com.MAVLink.Messages.MAVLinkMessage;
5:  import com.MAVLink.Messages.MAVLinkPayload;
6:  import com.MAVLink.Messages.MAVLinkPacket;
7:  //import android.util.Log;
8:
9:  /**
10:   *
11:   */
12:  public class msg_request_data_stream extends MAVLinkMessage{
13:
14:      public static final int MAVLINK_MSG_ID_REQUEST_DATA_STREAM = 66;
15:      public static final int MAVLINK_MSG_LENGTH = 6;
16:      private static final long serialVersionUID = MAVLINK_MSG_ID_REQUEST_DATA_S
TREAM;
17:
18:
19:      /**
20:       * The requested interval between two messages of this type
21:       */
22:      public short req_message_rate;
23:      /**
24:       * The target requested to send the message stream.
25:       */
26:      public byte target_system;
27:      /**
28:       * The target requested to send the message stream.
29:       */
30:      public byte target_component;
31:      /**
32:       * The ID of the requested data stream
33:       */
34:      public byte req_stream_id;
35:      /**
36:       * 1 to start sending, 0 to stop sending.
37:       */
38:      public byte start_stop;
39:
40:      /**
41:       * Generates the payload for a mavlink message for a message of this type
42:       * @return
43:       */
44:      public MAVLinkPacket pack(){
45:          MAVLinkPacket packet = new MAVLinkPacket();
46:          packet.len = MAVLINK_MSG_LENGTH;
47:          packet.sysid = 255;
48:          packet.compid = 190;
49:          packet.msgid = MAVLINK_MSG_ID_REQUEST_DATA_STREAM;
50:          packet.payload.putShort(req_message_rate);
51:          packet.payload.putByte(target_system);
52:          packet.payload.putByte(target_component);
53:          packet.payload.putByte(req_stream_id);
54:          packet.payload.putByte(start_stop);
55:          return packet;
56:      }
57:
58:      /**
59:       * Decode a request_data_stream message into this class fields
60:       *
61:       * @param payload The message to decode
62:       */
63:      public void unpack(MAVLinkPayload payload) {
64:          payload.resetIndex();
65:          req_message_rate = payload.getShort();
66:          target_system = payload.getByte();

```

```

67:          target_component = payload.getByte();
68:          req_stream_id = payload.getByte();
69:          start_stop = payload.getByte();
70:      }
71:
72:      /**
73:       * Constructor for a new message, just initializes the msgid
74:       */
75:      public msg_request_data_stream(){
76:          msgid = MAVLINK_MSG_ID_REQUEST_DATA_STREAM;
77:      }
78:
79:      /**
80:       * Constructor for a new message, initializes the message with the payload
81:       * from a mavlink packet
82:       */
83:      /**
84:       * public msg_request_data_stream(MAVLinkPacket mavLinkPacket){
85:          this.sysid = mavLinkPacket.sysid;
86:          this.compid = mavLinkPacket.compid;
87:          this.msgid = MAVLINK_MSG_ID_REQUEST_DATA_STREAM;
88:          unpack(mavLinkPacket.payload);
89:          //Log.d("MAVLink", "REQUEST_DATA_STREAM");
90:          //Log.d("MAVLINK_MSG_ID_REQUEST_DATA_STREAM", toString());
91:      }
92:
93:
94:      /**
95:       * Returns a string with the MSG name and data
96:       */
97:      public String toString(){
98:          return "MAVLINK_MSG_ID_REQUEST_DATA_STREAM -"+" req_message_rate:"+req_mes
sage_rate+" target_system:"+target_system+" target_component:"+target_component+" req_str
eam_id:"+req_stream_id+" start_stop:"+start_stop+"";
99:      }
100: }

```



```

1: // MESSAGE ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Setpoint in rollspeed, pitchspeed, yaw speed currently active on the system.
11:  */
12: public class msg_roll_pitch_yaw_speed_thrust_setpoint extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT
T = 59;
15:     public static final int MAVLINK_MSG_LENGTH = 20;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_ROLL_PITCH_YAW
_SPEED_THRUST_SETPOINT;
17:
18:
19:     /**
20:     * Timestamp in milliseconds since system boot
21:     */
22:     public int time_boot_ms;
23:     /**
24:     * Desired roll angular speed in rad/s
25:     */
26:     public float roll_speed;
27:     /**
28:     * Desired pitch angular speed in rad/s
29:     */
30:     public float pitch_speed;
31:     /**
32:     * Desired yaw angular speed in rad/s
33:     */
34:     public float yaw_speed;
35:     /**
36:     * Collective thrust, normalized to 0 .. 1
37:     */
38:     public float thrust;
39:
40:     /**
41:     * Generates the payload for a mavlink message for a message of this type
42:     * @return
43:     */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compuid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT
;
50:
51:         packet.payload.putInt(time_boot_ms);
52:         packet.payload.putFloat(roll_speed);
53:         packet.payload.putFloat(pitch_speed);
54:         packet.payload.putFloat(yaw_speed);
55:         packet.payload.putFloat(thrust);
56:         return packet;
57:
58:     /**
59:     * Decode a roll_pitch_yaw_speed_thrust_setpoint message into this class field
S
60:     *
61:     * @param payload The message to decode
62:     */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         time_boot_ms = payload.getInt();
66:         roll_speed = payload.getFloat();
67:         pitch_speed = payload.getFloat();
68:         yaw_speed = payload.getFloat();
69:         thrust = payload.getFloat();
70:     }
71:
72:     /**
73:     * Constructor for a new message, just initializes the msgid
74:     */
75:     public msg_roll_pitch_yaw_speed_thrust_setpoint(){
76:         msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT;
77:     }
78:
79:     /**
80:     * Constructor for a new message, initializes the message with the payload
81:     * from a mavlink packet
82:     */
83:     /**
84:     public msg_roll_pitch_yaw_speed_thrust_setpoint(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compuid = mavLinkPacket.compuid;
87:         this.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT");
90:         //Log.d("MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT", toString())
;
91:     }
92:
93:
94:     /**
95:     * Returns a string with the MSG name and data
96:     */
97:     public String toString(){
98:         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:"+ya
w_speed+" thrust:"+thrust+"";
99:     }
100: }

```



```

1: // MESSAGE ROLL_PITCH_YAW_THRUST_SETPOINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Setpoint in roll, pitch, yaw currently active on the system.
11:  */
12: public class msg_roll_pitch_yaw_thrust_setpoint extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT = 58
;
15:     public static final int MAVLINK_MSG_LENGTH = 20;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_ROLL_PITCH_YAW
_THRUST_SETPOINT;
17:
18:
19:     /**
20:     * Timestamp in milliseconds since system boot
21:     */
22:     public int time_boot_ms;
23:     /**
24:     * Desired roll angle in radians
25:     */
26:     public float roll;
27:     /**
28:     * Desired pitch angle in radians
29:     */
30:     public float pitch;
31:     /**
32:     * Desired yaw angle in radians
33:     */
34:     public float yaw;
35:     /**
36:     * Collective thrust, normalized to 0 .. 1
37:     */
38:     public float thrust;
39:
40:     /**
41:     * Generates the payload for a mavlink message for a message of this type
42:     * @return
43:     */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compuid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT;
50:         packet.payload.putInt(time_boot_ms);
51:         packet.payload.putFloat(roll);
52:         packet.payload.putFloat(pitch);
53:         packet.payload.putFloat(yaw);
54:         packet.payload.putFloat(thrust);
55:         return packet;
56:     }
57:
58:     /**
59:     * Decode a roll_pitch_yaw_thrust_setpoint message into this class fields
60:     *
61:     * @param payload The message to decode
62:     */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         time_boot_ms = payload.getInt();
66:
67:         roll = payload.getFloat();
68:         pitch = payload.getFloat();
69:         yaw = payload.getFloat();
70:         thrust = payload.getFloat();
71:     }
72:
73:     /**
74:     * Constructor for a new message, just initializes the msgid
75:     */
76:     public msg_roll_pitch_yaw_thrust_setpoint(){
77:         msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT;
78:     }
79:
80:     /**
81:     * Constructor for a new message, initializes the message with the payload
82:     * from a mavlink packet
83:     */
84:     public msg_roll_pitch_yaw_thrust_setpoint(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compuid = mavLinkPacket.compuid;
87:         this.msgid = MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "ROLL_PITCH_YAW_THRUST_SETPOINT");
90:         //Log.d("MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT", toString());
91:     }
92:
93:
94:     /**
95:     * Returns a string with the MSG name and data
96:     */
97:     public String toString(){
98:         return "MAVLINK_MSG_ID_ROLL_PITCH_YAW_THRUST_SETPOINT -"+" time_boot_ms:"+
time_boot_ms+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+" thrust:"+thrust+"";
99:     }
100: }

```



```

1: // MESSAGE SAFETY_ALLOWED_AREA PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Read out the safety zone the MAV currently assumes.
11:  */
12: public class msg_safety_allowed_area extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA = 55;
15:     public static final int MAVLINK_MSG_LENGTH = 25;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SAFETY_ALLOWED
AREA;
17:
18:
19:     /**
20:      * x position 1 / Latitude 1
21:      */
22:     public float plx;
23:     /**
24:      * y position 1 / Longitude 1
25:      */
26:     public float ply;
27:     /**
28:      * z position 1 / Altitude 1
29:      */
30:     public float plz;
31:     /**
32:      * x position 2 / Latitude 2
33:      */
34:     public float p2x;
35:     /**
36:      * y position 2 / Longitude 2
37:      */
38:     public float p2y;
39:     /**
40:      * z position 2 / Altitude 2
41:      */
42:     public float p2z;
43:     /**
44:      * 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.
45:      */
46:     public byte frame;
47:
48:     /**
49:      * Generates the payload for a mavlink message for a message of this type
50:      * @return
51:      */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compuid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA;
58:         packet.payload.putFloat(plx);
59:         packet.payload.putFloat(ply);
60:         packet.payload.putFloat(plz);
61:         packet.payload.putFloat(p2x);
62:         packet.payload.putFloat(p2y);
63:         packet.payload.putFloat(p2z);
64:         packet.payload.putByte(frame);
65:         return packet;

```

```

66:     }
67:
68:     /**
69:      * Decode a safety_allowed_area message into this class fields
70:      *
71:      * @param payload The message to decode
72:      */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         plx = payload.getFloat();
76:         ply = payload.getFloat();
77:         plz = payload.getFloat();
78:         p2x = payload.getFloat();
79:         p2y = payload.getFloat();
80:         p2z = payload.getFloat();
81:         frame = payload.getByte();
82:     }
83:
84:     /**
85:      * Constructor for a new message, just initializes the msgid
86:      */
87:     public msg_safety_allowed_area(){
88:         msgid = MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA;
89:     }
90:
91:     /**
92:      * Constructor for a new message, initializes the message with the payload
93:      * from a mavlink packet
94:      *
95:      */
96:     public msg_safety_allowed_area(MAVLinkPacket mavLinkPacket){
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compuid = mavLinkPacket.compuid;
99:         this.msgid = MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "SAFETY_ALLOWED_AREA");
102:         //Log.d("MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA", toString());
103:     }
104:
105:
106:     /**
107:      * Returns a string with the MSG name and data
108:      */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA -"+" plx:"+plx+" ply:"+ply+" pl
z:"+plz+" p2x:"+p2x+" p2y:"+p2y+" p2z:"+p2z+" frame:"+frame+"";
111:     }
112: }

```



```

1: // MESSAGE SAFETY_SET_ALLOWED_AREA PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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.
11:  */
12: public class msg_safety_set_allowed_area extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA = 54;
15:     public static final int MAVLINK_MSG_LENGTH = 27;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;
17:
18:     /**
19:      * x position 1 / Latitude 1
20:      */
21:     public float plx;
22:     /**
23:      * y position 1 / Longitude 1
24:      */
25:     public float ply;
26:     /**
27:      * z position 1 / Altitude 1
28:      */
29:     public float plz;
30:     /**
31:      * x position 2 / Latitude 2
32:      */
33:     public float p2x;
34:     /**
35:      * y position 2 / Longitude 2
36:      */
37:     public float p2y;
38:     /**
39:      * z position 2 / Altitude 2
40:      */
41:     public float p2z;
42:     /**
43:      * System ID
44:      */
45:     public byte target_system;
46:     /**
47:      * Component ID
48:      */
49:     public byte target_component;
50:     /**
51:      * 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.
52:      */
53:     public byte frame;
54:
55:     /**
56:      * Generates the payload for a mavlink message for a message of this type
57:      * @return
58:      */
59:     public MAVLinkPacket pack(){
60:         MAVLinkPacket packet = new MAVLinkPacket();
61:         packet.len = MAVLINK_MSG_LENGTH;
62:         packet.sysid = 255;
63:
64:         packet.compuid = 190;
65:         packet.msgid = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;
66:         packet.payload.putFloat(plx);
67:         packet.payload.putFloat(ply);
68:         packet.payload.putFloat(plz);
69:         packet.payload.putFloat(p2x);
70:         packet.payload.putFloat(p2y);
71:         packet.payload.putFloat(p2z);
72:         packet.payload.putByte(target_system);
73:         packet.payload.putByte(target_component);
74:         packet.payload.putByte(frame);
75:         return packet;
76:     }
77:
78:     /**
79:      * Decode a safety_set_allowed_area message into this class fields
80:      *
81:      * @param payload The message to decode
82:      */
83:     public void unpack(MAVLinkPayload payload) {
84:         payload.resetIndex();
85:         plx = payload.getFloat();
86:         ply = payload.getFloat();
87:         plz = payload.getFloat();
88:         p2x = payload.getFloat();
89:         p2y = payload.getFloat();
90:         p2z = payload.getFloat();
91:         target_system = payload.getByte();
92:         target_component = payload.getByte();
93:         frame = payload.getByte();
94:     }
95:
96:     /**
97:      * Constructor for a new message, just initializes the msgid
98:      */
99:     public msg_safety_set_allowed_area(){
100:         msgid = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;
101:     }
102:
103:     /**
104:      * Constructor for a new message, initializes the message with the payload from a mavlink packet
105:      *
106:      */
107:     public msg_safety_set_allowed_area(MAVLinkPacket mavLinkPacket){
108:         this.sysid = mavLinkPacket.sysid;
109:         this.compuid = mavLinkPacket.compuid;
110:         this.msgid = MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA;
111:         unpack(mavLinkPacket.payload);
112:         //Log.d("MAVLink", "SAFETY_SET_ALLOWED_AREA");
113:         //Log.d("MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA", toString());
114:     }
115:
116:     /**
117:      * Returns a string with the MSG name and data
118:      */
119:     public String toString(){
120:         return "MAVLINK_MSG_ID_SAFETY_SET_ALLOWED_AREA -"+" plx:"+plx+" ply:"+ply+" plz:"+plz+" p2x:"+p2x+" p2y:"+p2y+" p2z:"+p2z+" target_system:"+target_system+" target_component:"+target_component+" frame:"+frame+"";
121:     }
122: }

```



```

1: // MESSAGE SCALED_IMU PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * The RAW IMU readings for the usual 9DOF sensor setup. This message should contain the scaled values to the described units
11:  */
12: public class msg_scaled_imu extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SCALED_IMU = 26;
15:     public static final int MAVLINK_MSG_LENGTH = 22;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SCALED_IMU;
17:
18:
19:     /**
20:      * Timestamp (milliseconds since system boot)
21:      */
22:     public int time_boot_ms;
23:     /**
24:      * X acceleration (mg)
25:      */
26:     public short xacc;
27:     /**
28:      * Y acceleration (mg)
29:      */
30:     public short yacc;
31:     /**
32:      * Z acceleration (mg)
33:      */
34:     public short zacc;
35:     /**
36:      * Angular speed around X axis (millirad /sec)
37:      */
38:     public short xgyro;
39:     /**
40:      * Angular speed around Y axis (millirad /sec)
41:      */
42:     public short ygyro;
43:     /**
44:      * Angular speed around Z axis (millirad /sec)
45:      */
46:     public short zgyro;
47:     /**
48:      * X Magnetic field (milli tesla)
49:      */
50:     public short xmag;
51:     /**
52:      * Y Magnetic field (milli tesla)
53:      */
54:     public short ymag;
55:     /**
56:      * Z Magnetic field (milli tesla)
57:      */
58:     public short zmag;
59:
60:     /**
61:      * Generates the payload for a mavlink message for a message of this type
62:      * @return
63:      */
64:     public MAVLinkPacket pack(){
65:         MAVLinkPacket packet = new MAVLinkPacket();
66:         packet.len = MAVLINK_MSG_LENGTH;

```

```

67:         packet.sysid = 255;
68:         packet.compid = 190;
69:         packet.msgid = MAVLINK_MSG_ID_SCALED_IMU;
70:         packet.payload.putInt(time_boot_ms);
71:         packet.payload.putShort(xacc);
72:         packet.payload.putShort(yacc);
73:         packet.payload.putShort(zacc);
74:         packet.payload.putShort(xgyro);
75:         packet.payload.putShort(ygyro);
76:         packet.payload.putShort(zgyro);
77:         packet.payload.putShort(xmag);
78:         packet.payload.putShort(ymag);
79:         packet.payload.putShort(zmag);
80:         return packet;
81:     }
82:
83:     /**
84:      * Decode a scaled_imu message into this class fields
85:      *
86:      * @param payload The message to decode
87:      */
88:     public void unpack(MAVLinkPayload payload) {
89:         payload.resetIndex();
90:         time_boot_ms = payload.getInt();
91:         xacc = payload.getShort();
92:         yacc = payload.getShort();
93:         zacc = payload.getShort();
94:         xgyro = payload.getShort();
95:         ygyro = payload.getShort();
96:         zgyro = payload.getShort();
97:         xmag = payload.getShort();
98:         ymag = payload.getShort();
99:         zmag = payload.getShort();
100:     }
101:
102:     /**
103:      * Constructor for a new message, just initializes the msgid
104:      */
105:     public msg_scaled_imu(){
106:         msgid = MAVLINK_MSG_ID_SCALED_IMU;
107:     }
108:
109:     /**
110:      * Constructor for a new message, initializes the message with the payload
111:      * from a mavlink packet
112:      *
113:      */
114:     public msg_scaled_imu(MAVLinkPacket mavLinkPacket){
115:         this.sysid = mavLinkPacket.sysid;
116:         this.compid = mavLinkPacket.compid;
117:         this.msgid = MAVLINK_MSG_ID_SCALED_IMU;
118:         unpack(mavLinkPacket.payload);
119:         //Log.d("MAVLink", "SCALED_IMU");
120:         //Log.d("MAVLINK_MSG_ID_SCALED_IMU", toString());
121:     }
122:
123:
124:     /**
125:      * Returns a string with the MSG name and data
126:      */
127:     public String toString(){
128:         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+"";
129:     }
130: }

```



```

1: // MESSAGE SCALED_PRESSURE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The pressure readings for the typical setup of one absolute and differential pressure sensor. The units are as specified in each field.
11: */
12: public class msg_scaled_pressure extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SCALED_PRESSURE = 29;
15:     public static final int MAVLINK_MSG_LENGTH = 14;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SCALED_PRESSUR
E;
17:
18:
19:     /**
20:     * Timestamp (milliseconds since system boot)
21:     */
22:     public int time_boot_ms;
23:     /**
24:     * Absolute pressure (hectopascal)
25:     */
26:     public float press_abs;
27:     /**
28:     * Differential pressure 1 (hectopascal)
29:     */
30:     public float press_diff;
31:     /**
32:     * Temperature measurement (0.01 degrees celsius)
33:     */
34:     public short temperature;
35:
36:     /**
37:     * Generates the payload for a mavlink message for a message of this type
38:     * @return
39:     */
40:     public MAVLinkPacket pack(){
41:         MAVLinkPacket packet = new MAVLinkPacket();
42:         packet.len = MAVLINK_MSG_LENGTH;
43:         packet.sysid = 255;
44:         packet.compuid = 190;
45:         packet.msgid = MAVLINK_MSG_ID_SCALED_PRESSURE;
46:         packet.payload.putInt(time_boot_ms);
47:         packet.payload.putFloat(press_abs);
48:         packet.payload.putFloat(press_diff);
49:         packet.payload.putShort(temperature);
50:         return packet;
51:     }
52:
53:     /**
54:     * Decode a scaled_pressure message into this class fields
55:     *
56:     * @param payload The message to decode
57:     */
58:     public void unpack(MAVLinkPayload payload) {
59:         payload.resetIndex();
60:         time_boot_ms = payload.getInt();
61:         press_abs = payload.getFloat();
62:         press_diff = payload.getFloat();
63:         temperature = payload.getShort();
64:     }
65:

```

```

66:     /**
67:     * Constructor for a new message, just initializes the msgid
68:     */
69:     public msg_scaled_pressure(){
70:         msgid = MAVLINK_MSG_ID_SCALED_PRESSURE;
71:     }
72:
73:     /**
74:     * Constructor for a new message, initializes the message with the payload
75:     * from a mavlink packet
76:     */
77:     public msg_scaled_pressure(MAVLinkPacket mavLinkPacket){
78:         this.sysid = mavLinkPacket.sysid;
79:         this.compuid = mavLinkPacket.compuid;
80:         this.msgid = MAVLINK_MSG_ID_SCALED_PRESSURE;
81:         unpack(mavLinkPacket.payload);
82:         //Log.d("MAVLink", "SCALED_PRESSURE");
83:         //Log.d("MAVLINK_MSG_ID_SCALED_PRESSURE", toString());
84:     }
85:
86:
87:
88:     /**
89:     * Returns a string with the MSG name and data
90:     */
91:     public String toString(){
92:         return "MAVLINK_MSG_ID_SCALED_PRESSURE -" + time_boot_ms + ":" + time_boot_ms + "
press_abs:" + press_abs + " press_diff:" + press_diff + " temperature:" + temperature + "";
93:     }
94: }

```



```

1: // MESSAGE SENSOR_OFFSETS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Offsets and calibrations values for hardware
11:  * sensors. This makes it easier to debug the calibration process.
12:  */
13: public class msg_sensor_offsets extends MAVLinkMessage{
14:
15:     public static final int MAVLINK_MSG_ID_SENSOR_OFFSETS = 150;
16:     public static final int MAVLINK_MSG_LENGTH = 42;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_SENSOR_OFFSETS
;
18:
19:
20:     /**
21:      * magnetic declination (radians)
22:      */
23:     public float mag_declination;
24:     /**
25:      * raw pressure from barometer
26:      */
27:     public int raw_press;
28:     /**
29:      * raw temperature from barometer
30:      */
31:     public int raw_temp;
32:     /**
33:      * gyro X calibration
34:      */
35:     public float gyro_cal_x;
36:     /**
37:      * gyro Y calibration
38:      */
39:     public float gyro_cal_y;
40:     /**
41:      * gyro Z calibration
42:      */
43:     public float gyro_cal_z;
44:     /**
45:      * accel X calibration
46:      */
47:     public float accel_cal_x;
48:     /**
49:      * accel Y calibration
50:      */
51:     public float accel_cal_y;
52:     /**
53:      * accel Z calibration
54:      */
55:     public float accel_cal_z;
56:     /**
57:      * magnetometer X offset
58:      */
59:     public short mag_ofs_x;
60:     /**
61:      * magnetometer Y offset
62:      */
63:     public short mag_ofs_y;
64:     /**
65:      * magnetometer Z offset
66:      */

```

```

67:     public short mag_ofs_z;
68:
69:     /**
70:      * Generates the payload for a mavlink message for a message of this type
71:      * @return
72:      */
73:     public MAVLinkPacket pack(){
74:         MAVLinkPacket packet = new MAVLinkPacket();
75:         packet.len = MAVLINK_MSG_LENGTH;
76:         packet.sysid = 255;
77:         packet.compid = 190;
78:         packet.msgid = MAVLINK_MSG_ID_SENSOR_OFFSETS;
79:         packet.payload.putFloat(mag_declination);
80:         packet.payload.putInt(raw_press);
81:         packet.payload.putInt(raw_temp);
82:         packet.payload.putFloat(gyro_cal_x);
83:         packet.payload.putFloat(gyro_cal_y);
84:         packet.payload.putFloat(gyro_cal_z);
85:         packet.payload.putFloat(accel_cal_x);
86:         packet.payload.putFloat(accel_cal_y);
87:         packet.payload.putFloat(accel_cal_z);
88:         packet.payload.putShort(mag_ofs_x);
89:         packet.payload.putShort(mag_ofs_y);
90:         packet.payload.putShort(mag_ofs_z);
91:         return packet;
92:     }
93:
94:     /**
95:      * Decode a sensor_offsets message into this class fields
96:      *
97:      * @param payload The message to decode
98:      */
99:     public void unpack(MAVLinkPayload payload) {
100:         payload.resetIndex();
101:         mag_declination = payload.getFloat();
102:         raw_press = payload.getInt();
103:         raw_temp = payload.getInt();
104:         gyro_cal_x = payload.getFloat();
105:         gyro_cal_y = payload.getFloat();
106:         gyro_cal_z = payload.getFloat();
107:         accel_cal_x = payload.getFloat();
108:         accel_cal_y = payload.getFloat();
109:         accel_cal_z = payload.getFloat();
110:         mag_ofs_x = payload.getShort();
111:         mag_ofs_y = payload.getShort();
112:         mag_ofs_z = payload.getShort();
113:     }
114:
115:     /**
116:      * Constructor for a new message, just initializes the msgid
117:      */
118:     public msg_sensor_offsets(){
119:         msgid = MAVLINK_MSG_ID_SENSOR_OFFSETS;
120:     }
121:
122:     /**
123:      * Constructor for a new message, initializes the message with the payload
124:      * from a mavlink packet
125:      *
126:      */
127:     public msg_sensor_offsets(MAVLinkPacket mavLinkPacket){
128:         this.sysid = mavLinkPacket.sysid;
129:         this.compid = mavLinkPacket.compid;
130:         this.msgid = MAVLINK_MSG_ID_SENSOR_OFFSETS;
131:         unpack(mavLinkPacket.payload);
132:         //Log.d("MAVLink", "SENSOR_OFFSETS");
133:         //Log.d("MAVLINK_MSG_ID_SENSOR_OFFSETS", toString());

```

```

134:     }
135:
136:
137:     /**
138:      * Returns a string with the MSG name and data
139:      */
140:     public String toString(){
141:         return "MAVLINK_MSG_ID_SENSOR_OFFSETS -"+" mag_declination:"+mag_declinati
on+" 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_c
al_y+" accel_cal_z:"+accel_cal_z+" mag_ofs_x:"+mag_ofs_x+" mag_ofs_y:"+mag_ofs_y+" mag_of
s_z:"+mag_ofs_z+"";
142:     }
143: }

```

```

1: // MESSAGE SERVO_OUTPUT_RAW PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The RAW values of the servo outputs (for RC input from the remote, use the RC_CH
ANNELS messages). The standard PPM modulation is as follows: 1000 microseconds: 0%, 2000
microseconds: 100%.
11: */
12: public class msg_servo_output_raw extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SERVO_OUTPUT_RAW = 36;
15:     public static final int MAVLINK_MSG_LENGTH = 21;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SERVO_OUTPUT_R
AW;
17:
18:
19:     /**
20:     * Timestamp (microseconds since system boot)
21:     */
22:     public int time_usec;
23:     /**
24:     * Servo output 1 value, in microseconds
25:     */
26:     public short servo1_raw;
27:     /**
28:     * Servo output 2 value, in microseconds
29:     */
30:     public short servo2_raw;
31:     /**
32:     * Servo output 3 value, in microseconds
33:     */
34:     public short servo3_raw;
35:     /**
36:     * Servo output 4 value, in microseconds
37:     */
38:     public short servo4_raw;
39:     /**
40:     * Servo output 5 value, in microseconds
41:     */
42:     public short servo5_raw;
43:     /**
44:     * Servo output 6 value, in microseconds
45:     */
46:     public short servo6_raw;
47:     /**
48:     * Servo output 7 value, in microseconds
49:     */
50:     public short servo7_raw;
51:     /**
52:     * Servo output 8 value, in microseconds
53:     */
54:     public short servo8_raw;
55:     /**
56:     * Servo output port (set of 8 outputs = 1 port). Most MAVs will just use o
ne, but this allows to encode more than 8 servos.
57:     */
58:     public byte port;
59:
60:     /**
61:     * Generates the payload for a mavlink message for a message of this type
62:     * @return
63:     */

```

```

64:     public MAVLinkPacket pack(){
65:         MAVLinkPacket packet = new MAVLinkPacket();
66:         packet.len = MAVLINK_MSG_LENGTH;
67:         packet.sysid = 255;
68:         packet.compid = 190;
69:         packet.msgid = MAVLINK_MSG_ID_SERVO_OUTPUT_RAW;
70:         packet.payload.putInt(time_usec);
71:         packet.payload.putShort(servo1_raw);
72:         packet.payload.putShort(servo2_raw);
73:         packet.payload.putShort(servo3_raw);
74:         packet.payload.putShort(servo4_raw);
75:         packet.payload.putShort(servo5_raw);
76:         packet.payload.putShort(servo6_raw);
77:         packet.payload.putShort(servo7_raw);
78:         packet.payload.putShort(servo8_raw);
79:         packet.payload.putByte(port);
80:         return packet;
81:     }
82:
83:     /**
84:     * Decode a servo_output_raw message into this class fields
85:     *
86:     * @param payload The message to decode
87:     */
88:     public void unpack(MAVLinkPayload payload) {
89:         payload.resetIndex();
90:         time_usec = payload.getInt();
91:         servo1_raw = payload.getShort();
92:         servo2_raw = payload.getShort();
93:         servo3_raw = payload.getShort();
94:         servo4_raw = payload.getShort();
95:         servo5_raw = payload.getShort();
96:         servo6_raw = payload.getShort();
97:         servo7_raw = payload.getShort();
98:         servo8_raw = payload.getShort();
99:         port = payload.getByte();
100:     }
101:
102:     /**
103:     * Constructor for a new message, just initializes the msgid
104:     */
105:     public msg_servo_output_raw(){
106:         msgid = MAVLINK_MSG_ID_SERVO_OUTPUT_RAW;
107:     }
108:
109:     /**
110:     * Constructor for a new message, initializes the message with the payload
111:     * from a mavlink packet
112:     *
113:     */
114:     public msg_servo_output_raw(MAVLinkPacket mavLinkPacket){
115:         this.sysid = mavLinkPacket.sysid;
116:         this.compid = mavLinkPacket.compid;
117:         this.msgid = MAVLINK_MSG_ID_SERVO_OUTPUT_RAW;
118:         unpack(mavLinkPacket.payload);
119:         //Log.d("MAVLink", "SERVO_OUTPUT_RAW");
120:         //Log.d("MAVLINK_MSG_ID_SERVO_OUTPUT_RAW", toString());
121:     }
122:
123:
124:     /**
125:     * Returns a string with the MSG name and data
126:     */
127:     public String toString(){
128:         return "MAVLINK_MSG_ID_SERVO_OUTPUT_RAW -"+" time_usec:"+time_usec+" servo
1_raw:"+servo1_raw+" servo2_raw:"+servo2_raw+" servo3_raw:"+servo3_raw+" servo4_raw:"+ser
vo4_raw+" servo5_raw:"+servo5_raw+" servo6_raw:"+servo6_raw+" servo7_raw:"+servo7_raw+" s

```

./com/MAVLink/Messages/ardupilotmega/msg_servo_output_raw.java

Fri Oct 25 14:10:51 2013

2

```
ervo8_raw:"+servo8_raw+" port:"+port+"";
129:    }
130: }
```

```

1: // MESSAGE SETPOINT_6DOF PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set the 6 DOF setpoint for a attitude and position controller.
11:  */
12: public class msg_setpoint_6dof extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SETPOINT_6DOF = 149;
15:     public static final int MAVLINK_MSG_LENGTH = 25;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SETPOINT_6DOF;
17:
18:
19:     /**
20:      * Translational Component in x
21:      */
22:     public float trans_x;
23:     /**
24:      * Translational Component in y
25:      */
26:     public float trans_y;
27:     /**
28:      * Translational Component in z
29:      */
30:     public float trans_z;
31:     /**
32:      * Rotational Component in x
33:      */
34:     public float rot_x;
35:     /**
36:      * Rotational Component in y
37:      */
38:     public float rot_y;
39:     /**
40:      * Rotational Component in z
41:      */
42:     public float rot_z;
43:     /**
44:      * System ID
45:      */
46:     public byte target_system;
47:
48:     /**
49:      * Generates the payload for a mavlink message for a message of this type
50:      * @return
51:      */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compuid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_SETPOINT_6DOF;
58:         packet.payload.putFloat(trans_x);
59:         packet.payload.putFloat(trans_y);
60:         packet.payload.putFloat(trans_z);
61:         packet.payload.putFloat(rot_x);
62:         packet.payload.putFloat(rot_y);
63:         packet.payload.putFloat(rot_z);
64:         packet.payload.putByte(target_system);
65:         return packet;
66:     }
67:

```

```

68:     /**
69:      * Decode a setpoint_6dof message into this class fields
70:      *
71:      * @param payload The message to decode
72:      */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         trans_x = payload.getFloat();
76:         trans_y = payload.getFloat();
77:         trans_z = payload.getFloat();
78:         rot_x = payload.getFloat();
79:         rot_y = payload.getFloat();
80:         rot_z = payload.getFloat();
81:         target_system = payload.getByte();
82:     }
83:
84:     /**
85:      * Constructor for a new message, just initializes the msgid
86:      */
87:     public msg_setpoint_6dof(){
88:         msgid = MAVLINK_MSG_ID_SETPOINT_6DOF;
89:     }
90:
91:     /**
92:      * Constructor for a new message, initializes the message with the payload
93:      * from a mavlink packet
94:      *
95:      */
96:     public msg_setpoint_6dof(MAVLinkPacket mavLinkPacket){
97:         this.sysid = mavLinkPacket.sysid;
98:         this.compuid = mavLinkPacket.compuid;
99:         this.msgid = MAVLINK_MSG_ID_SETPOINT_6DOF;
100:         unpack(mavLinkPacket.payload);
101:         //Log.d("MAVLink", "SETPOINT_6DOF");
102:         //Log.d("MAVLINK_MSG_ID_SETPOINT_6DOF", toString());
103:     }
104:
105:
106:     /**
107:      * Returns a string with the MSG name and data
108:      */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_SETPOINT_6DOF -"+" trans_x:"+trans_x+" trans_y:"+tr
ans_y+" trans_z:"+trans_z+" rot_x:"+rot_x+" rot_y:"+rot_y+" rot_z:"+rot_z+" target_system
:"+target_system+"";
111:     }
112: }

```


./com/MAVLink/Messages/ardupilotmega/msg_setpoint_8dof.java

Fri Oct 25 14:10:51 2013

1

```
1: // MESSAGE SETPOINT_8DOF PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set the 8 DOF setpoint for a controller.
11:  */
12: public class msg_setpoint_8dof extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SETPOINT_8DOF = 148;
15:     public static final int MAVLINK_MSG_LENGTH = 33;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SETPOINT_8DOF;
17:
18:
19:     /**
20:      * Value 1
21:      */
22:     public float val1;
23:     /**
24:      * Value 2
25:      */
26:     public float val2;
27:     /**
28:      * Value 3
29:      */
30:     public float val3;
31:     /**
32:      * Value 4
33:      */
34:     public float val4;
35:     /**
36:      * Value 5
37:      */
38:     public float val5;
39:     /**
40:      * Value 6
41:      */
42:     public float val6;
43:     /**
44:      * Value 7
45:      */
46:     public float val7;
47:     /**
48:      * Value 8
49:      */
50:     public float val8;
51:     /**
52:      * System ID
53:      */
54:     public byte target_system;
55:
56:     /**
57:      * Generates the payload for a mavlink message for a message of this type
58:      * @return
59:      */
60:     public MAVLinkPacket pack(){
61:         MAVLinkPacket packet = new MAVLinkPacket();
62:         packet.len = MAVLINK_MSG_LENGTH;
63:         packet.sysid = 255;
64:         packet.compuid = 190;
65:         packet.msgid = MAVLINK_MSG_ID_SETPOINT_8DOF;
66:         packet.payload.putFloat(val1);
67:         packet.payload.putFloat(val2);
```

```
68:         packet.payload.putFloat(val3);
69:         packet.payload.putFloat(val4);
70:         packet.payload.putFloat(val5);
71:         packet.payload.putFloat(val6);
72:         packet.payload.putFloat(val7);
73:         packet.payload.putFloat(val8);
74:         packet.payload.putByte(target_system);
75:         return packet;
76:     }
77:
78:     /**
79:      * Decode a setpoint_8dof message into this class fields
80:      *
81:      * @param payload The message to decode
82:      */
83:     public void unpack(MAVLinkPayload payload) {
84:         payload.resetIndex();
85:         val1 = payload.getFloat();
86:         val2 = payload.getFloat();
87:         val3 = payload.getFloat();
88:         val4 = payload.getFloat();
89:         val5 = payload.getFloat();
90:         val6 = payload.getFloat();
91:         val7 = payload.getFloat();
92:         val8 = payload.getFloat();
93:         target_system = payload.getByte();
94:     }
95:
96:     /**
97:      * Constructor for a new message, just initializes the msgid
98:      */
99:     public msg_setpoint_8dof(){
100:         msgid = MAVLINK_MSG_ID_SETPOINT_8DOF;
101:     }
102:
103:     /**
104:      * Constructor for a new message, initializes the message with the payload
105:      * from a mavlink packet
106:      */
107:     public msg_setpoint_8dof(MAVLinkPacket mavLinkPacket){
108:         this.sysid = mavLinkPacket.sysid;
109:         this.compuid = mavLinkPacket.compuid;
110:         this.msgid = MAVLINK_MSG_ID_SETPOINT_8DOF;
111:         unpack(mavLinkPacket.payload);
112:         //Log.d("MAVLink", "SETPOINT_8DOF");
113:         //Log.d("MAVLINK_MSG_ID_SETPOINT_8DOF", toString());
114:     }
115:
116:
117:
118:     /**
119:      * Returns a string with the MSG name and data
120:      */
121:     public String toString(){
122:         return "MAVLINK_MSG_ID_SETPOINT_8DOF -"+" val1:"+val1+" val2:"+val2+" val3:
123:         "+val3+" val4:"+val4+" val5:"+val5+" val6:"+val6+" val7:"+val7+" val8:"+val8+" target_sy
124:         stem:"+target_system+"";
125:     }
126: }
```



```

1: // MESSAGE SET_GLOBAL_POSITION_SETPOINT_INT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set the current global position setpoint.
11:  */
12: public class msg_set_global_position_setpoint_int extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT =
53:
15:     public static final int MAVLINK_MSG_LENGTH = 15;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_GLOBAL_POS
TION_SETPOINT_INT;
17:
18:     /**
19:     * WGS84 Latitude position in degrees * 1E7
20:     */
21:     public int latitude;
22:     /**
23:     * WGS84 Longitude position in degrees * 1E7
24:     */
25:     public int longitude;
26:     /**
27:     * WGS84 Altitude in meters * 1000 (positive for up)
28:     */
29:     public int altitude;
30:     /**
31:     * Desired yaw angle in degrees * 100
32:     */
33:     public short yaw;
34:     /**
35:     * Coordinate frame - valid values are only MAV_FRAME_GLOBAL or MAV_FRAME_G
36:     LOCAL_RELATIVE_ALT
37:     */
38:     public byte coordinate_frame;
39:
40:     /**
41:     * Generates the payload for a mavlink message for a message of this type
42:     * @return
43:     */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT;
50:         packet.payload.putInt(latitude);
51:         packet.payload.putInt(longitude);
52:         packet.payload.putInt(altitude);
53:         packet.payload.putShort(yaw);
54:         packet.payload.putByte(coordinate_frame);
55:         return packet;
56:     }
57:
58:     /**
59:     * Decode a set_global_position_setpoint_int message into this class fields
60:     */
61:     * @param payload The message to decode
62:     */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         latitude = payload.getInt();
66:         longitude = payload.getInt();
67:         altitude = payload.getInt();
68:         yaw = payload.getShort();
69:         coordinate_frame = payload.getByte();
70:     }
71:
72:     /**
73:     * Constructor for a new message, just initializes the msgid
74:     */
75:     public msg_set_global_position_setpoint_int(){
76:         msgid = MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT;
77:     }
78:
79:     /**
80:     * Constructor for a new message, initializes the message with the payload
81:     * from a mavlink packet
82:     */
83:     /**
84:     public msg_set_global_position_setpoint_int(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compid = mavLinkPacket.compid;
87:         this.msgid = MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "SET_GLOBAL_POSITION_SETPOINT_INT");
90:         //Log.d("MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT", toString());
91:     }
92:
93:     /**
94:     * Returns a string with the MSG name and data
95:     */
96:     public String toString(){
97:         return "MAVLINK_MSG_ID_SET_GLOBAL_POSITION_SETPOINT_INT -"+" latitude:"+la
98:         titude+" longitude:"+longitude+" altitude:"+altitude+" yaw:"+yaw+" coordinate_frame:"+coo
99:         rdinate_frame+"";
100:     }

```



```

1: // MESSAGE SET_GPS_GLOBAL_ORIGIN PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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 indoor to outdoor.
11:  */
12: public class msg_set_gps_global_origin extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN = 48;
15:     public static final int MAVLINK_MSG_LENGTH = 13;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;
17:
18:
19:     /**
20:      * global position * 1E7
21:      */
22:     public int latitude;
23:     /**
24:      * global position * 1E7
25:      */
26:     public int longitude;
27:     /**
28:      * global position * 1000
29:      */
30:     public int altitude;
31:     /**
32:      * System ID
33:      */
34:     public byte target_system;
35:
36:     /**
37:      * Generates the payload for a mavlink message for a message of this type
38:      * @return
39:      */
40:     public MAVLinkPacket pack(){
41:         MAVLinkPacket packet = new MAVLinkPacket();
42:         packet.len = MAVLINK_MSG_LENGTH;
43:         packet.sysid = 255;
44:         packet.compid = 190;
45:         packet.msgid = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;
46:         packet.payload.putInt(latitude);
47:         packet.payload.putInt(longitude);
48:         packet.payload.putInt(altitude);
49:         packet.payload.putByte(target_system);
50:         return packet;
51:     }
52:
53:     /**
54:      * Decode a set_gps_global_origin message into this class fields
55:      *
56:      * @param payload The message to decode
57:      */
58:     public void unpack(MAVLinkPayload payload) {
59:         payload.resetIndex();
60:         latitude = payload.getInt();
61:         longitude = payload.getInt();
62:         altitude = payload.getInt();
63:         target_system = payload.getByte();
64:     }
65:
66:     /**
67:      * Constructor for a new message, just initializes the msgid
68:      */
69:     public msg_set_gps_global_origin(){
70:         msgid = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;
71:     }
72:
73:     /**
74:      * Constructor for a new message, initializes the message with the payload
75:      * from a mavlink packet
76:      */
77:     public msg_set_gps_global_origin(MAVLinkPacket mavLinkPacket){
78:         this.sysid = mavLinkPacket.sysid;
79:         this.compid = mavLinkPacket.compid;
80:         this.msgid = MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN;
81:         unpack(mavLinkPacket.payload);
82:         //Log.d("MAVLink", "SET_GPS_GLOBAL_ORIGIN");
83:         //Log.d("MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN", toString());
84:     }
85:
86:
87:
88:     /**
89:      * Returns a string with the MSG name and data
90:      */
91:     public String toString(){
92:         return "MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGIN -"+" latitude:"+latitude+" longitude:"+longitude+" altitude:"+altitude+" target_system:"+target_system+"";
93:     }
94: }

```



```

1: // MESSAGE SET_LOCAL_POSITION_SETPOINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set the setpoint for a local position controller. This is the position in local
11:  * coordinates the MAV should fly to. This message is sent by the path/MISSION planner to the
12:  * 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.
13:  */
14: public class msg_set_local_position_setpoint extends MAVLinkMessage{
15:     public static final int MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT = 50;
16:     public static final int MAVLINK_MSG_LENGTH = 19;
17:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;
18:
19:     /**
20:      * x position
21:      */
22:     public float x;
23:     /**
24:      * y position
25:      */
26:     public float y;
27:     /**
28:      * z position
29:      */
30:     public float z;
31:     /**
32:      * Desired yaw angle
33:      */
34:     public float yaw;
35:     /**
36:      * System ID
37:      */
38:     public byte target_system;
39:     /**
40:      * Component ID
41:      */
42:     public byte target_component;
43:     /**
44:      * Coordinate frame - valid values are only MAV_FRAME_LOCAL_NED or MAV_FRAME_GLOBAL_INT
45:      */
46:     public byte coordinate_frame;
47:
48:     /**
49:      * Generates the payload for a mavlink message for a message of this type
50:      * @return
51:      */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;
58:         packet.payload.putFloat(x);
59:         packet.payload.putFloat(y);
60:         packet.payload.putFloat(z);
61:         packet.payload.putFloat(yaw);
62:         packet.payload.putByte(target_system);
63:
64:         packet.payload.putByte(target_component);
65:         return packet;
66:     }
67:
68:     /**
69:      * Decode a set_local_position_setpoint message into this class fields
70:      *
71:      * @param payload The message to decode
72:      */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         x = payload.getFloat();
76:         y = payload.getFloat();
77:         z = payload.getFloat();
78:         yaw = payload.getFloat();
79:         target_system = payload.getByte();
80:         target_component = payload.getByte();
81:         coordinate_frame = payload.getByte();
82:     }
83:
84:     /**
85:      * Constructor for a new message, just initializes the msgid
86:      */
87:     public msg_set_local_position_setpoint(){
88:         msgid = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;
89:     }
90:
91:     /**
92:      * Constructor for a new message, initializes the message with the payload
93:      * from a mavlink packet
94:      */
95:     public msg_set_local_position_setpoint(MAVLinkPacket mavLinkPacket){
96:         this.sysid = mavLinkPacket.sysid;
97:         this.compid = mavLinkPacket.compid;
98:         this.msgid = MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT;
99:         unpack(mavLinkPacket.payload);
100:         //Log.d("MAVLink", "SET_LOCAL_POSITION_SETPOINT");
101:         //Log.d("MAVLINK_MSG_ID_SET_LOCAL_POSITION_SETPOINT", toString());
102:     }
103:
104:     /**
105:      * Returns a string with the MSG name and data
106:      */
107:     public String toString(){
108:         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 + " ";
109:     }
110: }

```



```

1: // MESSAGE SET_MAG_OFFSETS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * set the magnetometer offsets
11:  */
12: public class msg_set_mag_offsets extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_MAG_OFFSETS = 151;
15:     public static final int MAVLINK_MSG_LENGTH = 8;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_MAG_OFFSET
S;
17:
18:
19:     /**
20:      * magnetometer X offset
21:      */
22:     public short mag_ofs_x;
23:     /**
24:      * magnetometer Y offset
25:      */
26:     public short mag_ofs_y;
27:     /**
28:      * magnetometer Z offset
29:      */
30:     public short mag_ofs_z;
31:     /**
32:      * System ID
33:      */
34:     public byte target_system;
35:     /**
36:      * Component ID
37:      */
38:     public byte target_component;
39:
40:     /**
41:      * Generates the payload for a mavlink message for a message of this type
42:      * @return
43:      */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_SET_MAG_OFFSETS;
50:         packet.payload.putShort(mag_ofs_x);
51:         packet.payload.putShort(mag_ofs_y);
52:         packet.payload.putShort(mag_ofs_z);
53:         packet.payload.putByte(target_system);
54:         packet.payload.putByte(target_component);
55:         return packet;
56:     }
57:
58:     /**
59:      * Decode a set_mag_offsets message into this class fields
60:      *
61:      * @param payload The message to decode
62:      */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         mag_ofs_x = payload.getShort();
66:         mag_ofs_y = payload.getShort();

```

```

67:         mag_ofs_z = payload.getShort();
68:         target_system = payload.getByte();
69:         target_component = payload.getByte();
70:     }
71:
72:     /**
73:      * Constructor for a new message, just initializes the msgid
74:      */
75:     public msg_set_mag_offsets(){
76:         msgid = MAVLINK_MSG_ID_SET_MAG_OFFSETS;
77:     }
78:
79:     /**
80:      * Constructor for a new message, initializes the message with the payload
81:      * from a mavlink packet
82:      */
83:     /**
84:     public msg_set_mag_offsets(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compid = mavLinkPacket.compid;
87:         this.msgid = MAVLINK_MSG_ID_SET_MAG_OFFSETS;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "SET_MAG_OFFSETS");
90:         //Log.d("MAVLINK_MSG_ID_SET_MAG_OFFSETS", toString());
91:     }
92:
93:
94:     /**
95:      * Returns a string with the MSG name and data
96:      */
97:     public String toString(){
98:         return "MAVLINK_MSG_ID_SET_MAG_OFFSETS -"+" mag_ofs_x:"+mag_ofs_x+" mag_of
s_y:"+mag_ofs_y+" mag_ofs_z:"+mag_ofs_z+" target_system:"+target_system+" target_componen
t:"+target_component+"";
99:     }
100: }

```



```

1: // MESSAGE SET_QUAD_MOTORS_SETPOINT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Setpoint in the four motor speeds
11:  */
12: public class msg_set_quad_motors_setpoint extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT = 60;
15:     public static final int MAVLINK_MSG_LENGTH = 9;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_QUAD_MOTOR
S_SETPOINT;
17:
18:
19:     /**
20:      * Front motor in + configuration, front left motor in x configuration
21:      */
22:     public short motor_front_nw;
23:     /**
24:      * Right motor in + configuration, front right motor in x configuration
25:      */
26:     public short motor_right_ne;
27:     /**
28:      * Back motor in + configuration, back right motor in x configuration
29:      */
30:     public short motor_back_se;
31:     /**
32:      * Left motor in + configuration, back left motor in x configuration
33:      */
34:     public short motor_left_sw;
35:     /**
36:      * System ID of the system that should set these motor commands
37:      */
38:     public byte target_system;
39:
40:     /**
41:      * Generates the payload for a mavlink message for a message of this type
42:      * @return
43:      */
44:     public MAVLinkPacket pack(){
45:         MAVLinkPacket packet = new MAVLinkPacket();
46:         packet.len = MAVLINK_MSG_LENGTH;
47:         packet.sysid = 255;
48:         packet.compid = 190;
49:         packet.msgid = MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT;
50:         packet.payload.putShort(motor_front_nw);
51:         packet.payload.putShort(motor_right_ne);
52:         packet.payload.putShort(motor_back_se);
53:         packet.payload.putShort(motor_left_sw);
54:         packet.payload.putByte(target_system);
55:         return packet;
56:     }
57:
58:     /**
59:      * Decode a set_quad_motors_setpoint message into this class fields
60:      *
61:      * @param payload The message to decode
62:      */
63:     public void unpack(MAVLinkPayload payload) {
64:         payload.resetIndex();
65:         motor_front_nw = payload.getShort();
66:         motor_right_ne = payload.getShort();
67:
68:         motor_back_se = payload.getShort();
69:         motor_left_sw = payload.getShort();
70:         target_system = payload.getByte();
71:     }
72:
73:     /**
74:      * Constructor for a new message, just initializes the msgid
75:      */
76:     public msg_set_quad_motors_setpoint(){
77:         msgid = MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT;
78:     }
79:
80:     /**
81:      * Constructor for a new message, initializes the message with the payload
82:      * from a mavlink packet
83:      */
84:     public msg_set_quad_motors_setpoint(MAVLinkPacket mavLinkPacket){
85:         this.sysid = mavLinkPacket.sysid;
86:         this.compid = mavLinkPacket.compid;
87:         this.msgid = MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT;
88:         unpack(mavLinkPacket.payload);
89:         //Log.d("MAVLink", "SET_QUAD_MOTORS_SETPOINT");
90:         //Log.d("MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT", toString());
91:     }
92:
93:
94:     /**
95:      * Returns a string with the MSG name and data
96:      */
97:     public String toString(){
98:         return "MAVLINK_MSG_ID_SET_QUAD_MOTORS_SETPOINT -"+" motor_front_nw:"+moto
r_front_nw+" motor_right_ne:"+motor_right_ne+" motor_back_se:"+motor_back_se+" motor_left
_sw:"+motor_left_sw+" target_system:"+target_system+"";
99:     }
100: }

```



```

1: // MESSAGE SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Setpoint for up to four quadrotors in a group / wing
11:  */
12: public class msg_set_quad_swarm_led_roll_pitch_yaw_thrust extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST = 63;
15:     public static final int MAVLINK_MSG_LENGTH = 46;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;
17:
18:     /**
19:     * Desired roll angle in radians +-PI (+-32767)
20:     */
21:     public short roll[] = new short[4];
22:     /**
23:     * Desired pitch angle in radians +-PI (+-32767)
24:     */
25:     public short pitch[] = new short[4];
26:     /**
27:     * Desired yaw angle in radians, scaled to int16 +-PI (+-32767)
28:     */
29:     public short yaw[] = new short[4];
30:     /**
31:     * Collective thrust, scaled to uint16 (0..65535)
32:     */
33:     public short thrust[] = new short[4];
34:     /**
35:     * ID of the quadrotor group (0 - 255, up to 256 groups supported)
36:     */
37:     public byte group;
38:     /**
39:     * ID of the flight mode (0 - 255, up to 256 modes supported)
40:     */
41:     public byte mode;
42:     /**
43:     * RGB red channel (0-255)
44:     */
45:     public byte led_red[] = new byte[4];
46:     /**
47:     * RGB green channel (0-255)
48:     */
49:     public byte led_blue[] = new byte[4];
50:     /**
51:     * RGB blue channel (0-255)
52:     */
53:     public byte led_green[] = new byte[4];
54:     /**
55:     * Generates the payload for a mavlink message for a message of this type
56:     */
57:     @return
58:     */
59:     public MAVLinkPacket pack(){
60:         MAVLinkPacket packet = new MAVLinkPacket();
61:         packet.len = MAVLINK_MSG_LENGTH;
62:         packet.sysid = 255;
63:         packet.compuid = 190;
64:         packet.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;
65:
66:         for (int i = 0; i < roll.length; i++) {
67:             packet.payload.putShort(roll[i]);
68:         }
69:         for (int i = 0; i < pitch.length; i++) {
70:             packet.payload.putShort(pitch[i]);
71:         }
72:         for (int i = 0; i < yaw.length; i++) {
73:             packet.payload.putShort(yaw[i]);
74:         }
75:         for (int i = 0; i < thrust.length; i++) {
76:             packet.payload.putShort(thrust[i]);
77:         }
78:         packet.payload.putByte(group);
79:         packet.payload.putByte(mode);
80:         for (int i = 0; i < led_red.length; i++) {
81:             packet.payload.putByte(led_red[i]);
82:         }
83:         for (int i = 0; i < led_blue.length; i++) {
84:             packet.payload.putByte(led_blue[i]);
85:         }
86:         for (int i = 0; i < led_green.length; i++) {
87:             packet.payload.putByte(led_green[i]);
88:         }
89:         return packet;
90:     }
91:
92:     /**
93:     * Decode a set_quad_swarm_led_roll_pitch_yaw_thrust message into this class fields
94:     */
95:     @param payload The message to decode
96:     */
97:     public void unpack(MAVLinkPayload payload) {
98:         payload.resetIndex();
99:         for (int i = 0; i < roll.length; i++) {
100:             roll[i] = payload.getShort();
101:         }
102:         for (int i = 0; i < pitch.length; i++) {
103:             pitch[i] = payload.getShort();
104:         }
105:         for (int i = 0; i < yaw.length; i++) {
106:             yaw[i] = payload.getShort();
107:         }
108:         for (int i = 0; i < thrust.length; i++) {
109:             thrust[i] = payload.getShort();
110:         }
111:         group = payload.getByte();
112:         mode = payload.getByte();
113:         for (int i = 0; i < led_red.length; i++) {
114:             led_red[i] = payload.getByte();
115:         }
116:         for (int i = 0; i < led_blue.length; i++) {
117:             led_blue[i] = payload.getByte();
118:         }
119:         for (int i = 0; i < led_green.length; i++) {
120:             led_green[i] = payload.getByte();
121:         }
122:     }
123:
124:     /**
125:     * Constructor for a new message, just initializes the msgid
126:     */
127:     public msg_set_quad_swarm_led_roll_pitch_yaw_thrust(){
128:         msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;
129:     }
130:

```

```
131:     /**
132:      * Constructor for a new message, initializes the message with the payload
133:      * from a mavlink packet
134:      *
135:      */
136:     public msg_set_quad_swarm_led_roll_pitch_yaw_thrust(MAVLinkPacket mavLinkPacke
t){
137:         this.sysid = mavLinkPacket.sysid;
138:         this.compuid = mavLinkPacket.compuid;
139:         this.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST;
140:         unpack(mavLinkPacket.payload);
141:         //Log.d("MAVLink", "SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST");
142:         //Log.d("MAVLINK_MSG_ID_SET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST", toStrin
g());
143:     }
144:
145:
146:     /**
147:      * Returns a string with the MSG name and data
148:      */
149:     public String toString(){
150:         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+"";
151:     }
152: }
```

```

1: // MESSAGE SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Setpoint for up to four quadrotors in a group / wing
11:  */
12: public class msg_set_quad_swarm_roll_pitch_yaw_thrust extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST
T = 61;
15:     public static final int MAVLINK_MSG_LENGTH = 34;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_QUAD_SWARM
ROLL_PITCH_YAW_THRUST;
17:
18:
19:     /**
20:     * Desired roll angle in radians +-PI (+-32767)
21:     */
22:     public short roll[] = new short[4];
23:     /**
24:     * Desired pitch angle in radians +-PI (+-32767)
25:     */
26:     public short pitch[] = new short[4];
27:     /**
28:     * Desired yaw angle in radians, scaled to int16 +-PI (+-32767)
29:     */
30:     public short yaw[] = new short[4];
31:     /**
32:     * Collective thrust, scaled to uint16 (0..65535)
33:     */
34:     public short thrust[] = new short[4];
35:     /**
36:     * ID of the quadrotor group (0 - 255, up to 256 groups supported)
37:     */
38:     public byte group;
39:     /**
40:     * ID of the flight mode (0 - 255, up to 256 modes supported)
41:     */
42:     public byte mode;
43:
44:     /**
45:     * Generates the payload for a mavlink message for a message of this type
46:     * @return
47:     */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compuid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST
;
54:         for (int i = 0; i < roll.length; i++) {
55:             packet.payload.putShort(roll[i]);
56:         }
57:         for (int i = 0; i < pitch.length; i++) {
58:             packet.payload.putShort(pitch[i]);
59:         }
60:         for (int i = 0; i < yaw.length; i++) {
61:             packet.payload.putShort(yaw[i]);
62:         }
63:         for (int i = 0; i < thrust.length; i++) {
64:             packet.payload.putShort(thrust[i]);
65:         }
66:         packet.payload.putByte(group);
67:         packet.payload.putByte(mode);
68:         return packet;
69:     }
70:
71:     /**
72:     * Decode a set_quad_swarm_roll_pitch_yaw_thrust message into this class field
73:     */
74:     * @param payload The message to decode
75:     */
76:     public void unpack(MAVLinkPayload payload) {
77:         payload.resetIndex();
78:         for (int i = 0; i < roll.length; i++) {
79:             roll[i] = payload.getShort();
80:         }
81:         for (int i = 0; i < pitch.length; i++) {
82:             pitch[i] = payload.getShort();
83:         }
84:         for (int i = 0; i < yaw.length; i++) {
85:             yaw[i] = payload.getShort();
86:         }
87:         for (int i = 0; i < thrust.length; i++) {
88:             thrust[i] = payload.getShort();
89:         }
90:         group = payload.getByte();
91:         mode = payload.getByte();
92:     }
93:
94:     /**
95:     * Constructor for a new message, just initializes the msgid
96:     */
97:     public msg_set_quad_swarm_roll_pitch_yaw_thrust(){
98:         msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST;
99:     }
100:
101:     /**
102:     * Constructor for a new message, initializes the message with the payload
103:     * from a mavlink packet
104:     */
105:
106:     public msg_set_quad_swarm_roll_pitch_yaw_thrust(MAVLinkPacket mavLinkPacket){
107:         this.sysid = mavLinkPacket.sysid;
108:         this.compuid = mavLinkPacket.compuid;
109:         this.msgid = MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST;
110:         unpack(mavLinkPacket.payload);
111:         //Log.d("MAVLink", "SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST");
112:         //Log.d("MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST", toString())
;
113:     }
114:
115:
116:     /**
117:     * Returns a string with the MSG name and data
118:     */
119:     public String toString(){
120:         return "MAVLINK_MSG_ID_SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST -"+" roll:"+ro
ll+" pitch:"+pitch+" yaw:"+yaw+" thrust:"+thrust+" group:"+group+" mode:"+mode+"";
121:     }
122: }

```



```

1: // MESSAGE SET_ROLL_PITCH_YAW_SPEED_THRUST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set roll, pitch and yaw.
11:  */
12: public class msg_set_roll_pitch_yaw_speed_thrust extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST = 5
7;
15:     public static final int MAVLINK_MSG_LENGTH = 18;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_ROLL_PITCH
_YAW_SPEED_THRUST;
17:
18:
19:     /**
20:      * Desired roll angular speed in rad/s
21:      */
22:     public float roll_speed;
23:     /**
24:      * Desired pitch angular speed in rad/s
25:      */
26:     public float pitch_speed;
27:     /**
28:      * Desired yaw angular speed in rad/s
29:      */
30:     public float yaw_speed;
31:     /**
32:      * Collective thrust, normalized to 0 .. 1
33:      */
34:     public float thrust;
35:     /**
36:      * System ID
37:      */
38:     public byte target_system;
39:     /**
40:      * Component ID
41:      */
42:     public byte target_component;
43:
44:     /**
45:      * Generates the payload for a mavlink message for a message of this type
46:      * @return
47:      */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compuid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST;
54:         packet.payload.putFloat(roll_speed);
55:         packet.payload.putFloat(pitch_speed);
56:         packet.payload.putFloat(yaw_speed);
57:         packet.payload.putFloat(thrust);
58:         packet.payload.putByte(target_system);
59:         packet.payload.putByte(target_component);
60:         return packet;
61:     }
62:
63:     /**
64:      * Decode a set_roll_pitch_yaw_speed_thrust message into this class fields
65:
66:      * @param payload The message to decode
67:      */
68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         roll_speed = payload.getFloat();
71:         pitch_speed = payload.getFloat();
72:         yaw_speed = payload.getFloat();
73:         thrust = payload.getFloat();
74:         target_system = payload.getByte();
75:         target_component = payload.getByte();
76:     }
77:
78:     /**
79:      * Constructor for a new message, just initializes the msgid
80:      */
81:     public msg_set_roll_pitch_yaw_speed_thrust(){
82:         msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST;
83:     }
84:
85:     /**
86:      * Constructor for a new message, initializes the message with the payload
87:      * from a mavlink packet
88:      */
89:     public msg_set_roll_pitch_yaw_speed_thrust(MAVLinkPacket mavLinkPacket){
90:         this.sysid = mavLinkPacket.sysid;
91:         this.compuid = mavLinkPacket.compuid;
92:         this.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST;
93:         unpack(mavLinkPacket.payload);
94:         //Log.d("MAVLink", "SET_ROLL_PITCH_YAW_SPEED_THRUST");
95:         //Log.d("MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST", toString());
96:     }
97:
98:
99:
100:     /**
101:      * Returns a string with the MSG name and data
102:      */
103:     public String toString(){
104:         return "MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_SPEED_THRUST -"+" roll_speed:"+r
oll_speed+" pitch_speed:"+pitch_speed+" yaw_speed:"+yaw_speed+" thrust:"+thrust+" target_
system:"+target_system+" target_component:"+target_component+"";
105:     }
106: }

```



```

1: // MESSAGE SET_ROLL_PITCH_YAW_THRUST PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Set roll, pitch and yaw.
11:  */
12: public class msg_set_roll_pitch_yaw_thrust extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST = 56;
15:     public static final int MAVLINK_MSG_LENGTH = 18;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SET_ROLL_PITCH
_YAW_THRUST;
17:
18:
19:     /**
20:      * Desired roll angle in radians
21:      */
22:     public float roll;
23:     /**
24:      * Desired pitch angle in radians
25:      */
26:     public float pitch;
27:     /**
28:      * Desired yaw angle in radians
29:      */
30:     public float yaw;
31:     /**
32:      * Collective thrust, normalized to 0 .. 1
33:      */
34:     public float thrust;
35:     /**
36:      * System ID
37:      */
38:     public byte target_system;
39:     /**
40:      * Component ID
41:      */
42:     public byte target_component;
43:
44:     /**
45:      * Generates the payload for a mavlink message for a message of this type
46:      * @return
47:      */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST;
54:         packet.payload.putFloat(roll);
55:         packet.payload.putFloat(pitch);
56:         packet.payload.putFloat(yaw);
57:         packet.payload.putFloat(thrust);
58:         packet.payload.putByte(target_system);
59:         packet.payload.putByte(target_component);
60:         return packet;
61:     }
62:
63:     /**
64:      * Decode a set_roll_pitch_yaw_thrust message into this class fields
65:      *
66:      * @param payload The message to decode
67:      */
68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         roll = payload.getFloat();
71:         pitch = payload.getFloat();
72:         yaw = payload.getFloat();
73:         thrust = payload.getFloat();
74:         target_system = payload.getByte();
75:         target_component = payload.getByte();
76:     }
77:
78:     /**
79:      * Constructor for a new message, just initializes the msgid
80:      */
81:     public msg_set_roll_pitch_yaw_thrust(){
82:         msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST;
83:     }
84:
85:     /**
86:      * Constructor for a new message, initializes the message with the payload
87:      * from a mavlink packet
88:      */
89:     public msg_set_roll_pitch_yaw_thrust(MAVLinkPacket mavLinkPacket){
90:         this.sysid = mavLinkPacket.sysid;
91:         this.compid = mavLinkPacket.compid;
92:         this.msgid = MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST;
93:         unpack(mavLinkPacket.payload);
94:         //Log.d("MAVLink", "SET_ROLL_PITCH_YAW_THRUST");
95:         //Log.d("MAVLINK_MSG_ID_SET_ROLL_PITCH_YAW_THRUST", toString());
96:     }
97:
98:
99:
100:     /**
101:      * Returns a string with the MSG name and data
102:      */
103:     public String toString(){
104:         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+"";
105:     }
106: }

```



```

1: // MESSAGE SIMSTATE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Status of simulation environment, if used
11:  */
12: public class msg_simstate extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SIMSTATE = 164;
15:     public static final int MAVLINK_MSG_LENGTH = 44;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SIMSTATE;
17:
18:
19:     /**
20:      * Roll angle (rad)
21:      */
22:     public float roll;
23:     /**
24:      * Pitch angle (rad)
25:      */
26:     public float pitch;
27:     /**
28:      * Yaw angle (rad)
29:      */
30:     public float yaw;
31:     /**
32:      * X acceleration m/s/s
33:      */
34:     public float xacc;
35:     /**
36:      * Y acceleration m/s/s
37:      */
38:     public float yacc;
39:     /**
40:      * Z acceleration m/s/s
41:      */
42:     public float zacc;
43:     /**
44:      * Angular speed around X axis rad/s
45:      */
46:     public float xgyro;
47:     /**
48:      * Angular speed around Y axis rad/s
49:      */
50:     public float ygyro;
51:     /**
52:      * Angular speed around Z axis rad/s
53:      */
54:     public float zgyro;
55:     /**
56:      * Latitude in degrees
57:      */
58:     public float lat;
59:     /**
60:      * Longitude in degrees
61:      */
62:     public float lng;
63:
64:     /**
65:      * Generates the payload for a mavlink message for a message of this type
66:      * @return
67:      */

```

```

68:     public MAVLinkPacket pack(){
69:         MAVLinkPacket packet = new MAVLinkPacket();
70:         packet.len = MAVLINK_MSG_LENGTH;
71:         packet.sysid = 255;
72:         packet.compid = 190;
73:         packet.msgid = MAVLINK_MSG_ID_SIMSTATE;
74:         packet.payload.putFloat(roll);
75:         packet.payload.putFloat(pitch);
76:         packet.payload.putFloat(yaw);
77:         packet.payload.putFloat(xacc);
78:         packet.payload.putFloat(yacc);
79:         packet.payload.putFloat(zacc);
80:         packet.payload.putFloat(xgyro);
81:         packet.payload.putFloat(ygyro);
82:         packet.payload.putFloat(zgyro);
83:         packet.payload.putFloat(lat);
84:         packet.payload.putFloat(lng);
85:         return packet;
86:     }
87:
88:     /**
89:      * Decode a simstate message into this class fields
90:      *
91:      * @param payload The message to decode
92:      */
93:     public void unpack(MAVLinkPayload payload) {
94:         payload.resetIndex();
95:         roll = payload.getFloat();
96:         pitch = payload.getFloat();
97:         yaw = payload.getFloat();
98:         xacc = payload.getFloat();
99:         yacc = payload.getFloat();
100:         zacc = payload.getFloat();
101:         xgyro = payload.getFloat();
102:         ygyro = payload.getFloat();
103:         zgyro = payload.getFloat();
104:         lat = payload.getFloat();
105:         lng = payload.getFloat();
106:     }
107:
108:     /**
109:      * Constructor for a new message, just initializes the msgid
110:      */
111:     public msg_simstate(){
112:         msgid = MAVLINK_MSG_ID_SIMSTATE;
113:     }
114:
115:     /**
116:      * Constructor for a new message, initializes the message with the payload
117:      * from a mavlink packet
118:      */
119:     public msg_simstate(MAVLinkPacket mavLinkPacket){
120:         this.sysid = mavLinkPacket.sysid;
121:         this.compid = mavLinkPacket.compid;
122:         this.msgid = MAVLINK_MSG_ID_SIMSTATE;
123:         unpack(mavLinkPacket.payload);
124:         //Log.d("MAVLink", "SIMSTATE");
125:         //Log.d("MAVLINK_MSG_ID_SIMSTATE", toString());
126:     }
127:
128:
129:
130:     /**
131:      * Returns a string with the MSG name and data
132:      */
133:     public String toString(){
134:         return "MAVLINK_MSG_ID_SIMSTATE -"+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw

```

```
aw+" xacc:"+xacc+" yacc:"+yacc+" zacc:"+zacc+" xgyro:"+xgyro+" ygyro:"+ygyro+" zgyro:"+zgyro+" lat:"+lat+" lng:"+lng+"";
135:     }
136: }
```

```

1: // MESSAGE STATE_CORRECTION PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Corrects the systems state by adding an error correction term to the position an
d velocity, and by rotating the attitude by a correction angle.
11:  */
12: public class msg_state_correction extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_STATE_CORRECTION = 64;
15:     public static final int MAVLINK_MSG_LENGTH = 36;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_STATE_CORRECTI
ON;
17:
18:
19:     /**
20:     * x position error
21:     */
22:     public float xErr;
23:     /**
24:     * y position error
25:     */
26:     public float yErr;
27:     /**
28:     * z position error
29:     */
30:     public float zErr;
31:     /**
32:     * roll error (radians)
33:     */
34:     public float rollErr;
35:     /**
36:     * pitch error (radians)
37:     */
38:     public float pitchErr;
39:     /**
40:     * yaw error (radians)
41:     */
42:     public float yawErr;
43:     /**
44:     * x velocity
45:     */
46:     public float vxErr;
47:     /**
48:     * y velocity
49:     */
50:     public float vyErr;
51:     /**
52:     * z velocity
53:     */
54:     public float vzErr;
55:
56:     /**
57:     * Generates the payload for a mavlink message for a message of this type
58:     * @return
59:     */
60:     public MAVLinkPacket pack(){
61:         MAVLinkPacket packet = new MAVLinkPacket();
62:         packet.len = MAVLINK_MSG_LENGTH;
63:         packet.sysid = 255;
64:         packet.compuid = 190;
65:         packet.msgid = MAVLINK_MSG_ID_STATE_CORRECTION;

```

```

66:         packet.payload.putFloat(xErr);
67:         packet.payload.putFloat(yErr);
68:         packet.payload.putFloat(zErr);
69:         packet.payload.putFloat(rollErr);
70:         packet.payload.putFloat(pitchErr);
71:         packet.payload.putFloat(yawErr);
72:         packet.payload.putFloat(vxErr);
73:         packet.payload.putFloat(vyErr);
74:         packet.payload.putFloat(vzErr);
75:         return packet;
76:     }
77:
78:     /**
79:     * Decode a state_correction message into this class fields
80:     *
81:     * @param payload The message to decode
82:     */
83:     public void unpack(MAVLinkPayload payload) {
84:         payload.resetIndex();
85:         xErr = payload.getFloat();
86:         yErr = payload.getFloat();
87:         zErr = payload.getFloat();
88:         rollErr = payload.getFloat();
89:         pitchErr = payload.getFloat();
90:         yawErr = payload.getFloat();
91:         vxErr = payload.getFloat();
92:         vyErr = payload.getFloat();
93:         vzErr = payload.getFloat();
94:     }
95:
96:     /**
97:     * Constructor for a new message, just initializes the msgid
98:     */
99:     public msg_state_correction(){
100:         msgid = MAVLINK_MSG_ID_STATE_CORRECTION;
101:     }
102:
103:     /**
104:     * Constructor for a new message, initializes the message with the payload
105:     * from a mavlink packet
106:     */
107:
108:     public msg_state_correction(MAVLinkPacket mavLinkPacket){
109:         this.sysid = mavLinkPacket.sysid;
110:         this.compuid = mavLinkPacket.compuid;
111:         this.msgid = MAVLINK_MSG_ID_STATE_CORRECTION;
112:         unpack(mavLinkPacket.payload);
113:         //Log.d("MAVLink", "STATE_CORRECTION");
114:         //Log.d("MAVLINK_MSG_ID_STATE_CORRECTION", toString());
115:     }
116:
117:
118:     /**
119:     * Returns a string with the MSG name and data
120:     */
121:     public String toString(){
122:         return "MAVLINK_MSG_ID_STATE_CORRECTION -"+" xErr:"+xErr+" yErr:"+yErr+" z
Err:"+zErr+" rollErr:"+rollErr+" pitchErr:"+pitchErr+" yawErr:"+yawErr+" vxErr:"+vxErr+"
vyErr:"+vyErr+" vzErr:"+vzErr+"";
123:     }
124: }

```



```

1: // MESSAGE STATUSTEXT PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * 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 st
  atus and error messages. If implemented wisely, these messages are buffered on the MCU an
  d sent only at a limited rate (e.g. 10 Hz).
11:  */
12: public class msg_statustext extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_STATUSTEXT = 253;
15:     public static final int MAVLINK_MSG_LENGTH = 51;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_STATUSTEXT;
17:
18:
19:     /**
20:      * Severity of status. Relies on the definitions within RFC-5424. See enum
  MAV_SEVERITY.
21:      */
22:     public byte severity;
23:     /**
24:      * Status text message, without null termination character
25:      */
26:     public byte text[] = new byte[50];
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compId = 190;
37:         packet.msgid = MAVLINK_MSG_ID_STATUSTEXT;
38:         packet.payload.putByte(severity);
39:         for (int i = 0; i < text.length; i++) {
40:             packet.payload.putByte(text[i]);
41:         }
42:         return packet;
43:     }
44:
45:     /**
46:      * Decode a statustext message into this class fields
47:      *
48:      * @param payload The message to decode
49:      */
50:     public void unpack(MAVLinkPayload payload) {
51:         payload.resetIndex();
52:         severity = payload.getByte();
53:         for (int i = 0; i < text.length; i++) {
54:             text[i] = payload.getByte();
55:         }
56:     }
57:
58:     /**
59:      * Constructor for a new message, just initializes the msgid
60:      */
61:     public msg_statustext(){
62:         msgid = MAVLINK_MSG_ID_STATUSTEXT;
63:     }

```

```

64:
65:     /**
66:      * Constructor for a new message, initializes the message with the payload
67:      * from a mavlink packet
68:      */
69:     public msg_statustext(MAVLinkPacket mavLinkPacket){
70:         this.sysid = mavLinkPacket.sysid;
71:         this.compId = mavLinkPacket.compId;
72:         this.msgid = MAVLINK_MSG_ID_STATUSTEXT;
73:         unpack(mavLinkPacket.payload);
74:         //Log.d("MAVLink", "STATUSTEXT");
75:         //Log.d("MAVLINK_MSG_ID_STATUSTEXT", toString());
76:     }
77:
78:     /**
79:      * Sets the buffer of this message with a string, adds the necessary padding
80:      */
81:     public void setText(String str) {
82:         int len = Math.min(str.length(), 50);
83:         for (int i=0; i<len; i++) {
84:             text[i] = (byte) str.charAt(i);
85:         }
86:         for (int i=len; i<50; i++) {
87:             // padding for the rest of
  the buffer
88:             text[i] = 0;
89:         }
90:     }
91:
92:     /**
93:      * Gets the message, formatted as a string
94:      */
95:     public String getText() {
96:         String result = "";
97:         for (int i = 0; i < 50; i++) {
98:             if (text[i] != 0)
99:                 result = result + (char) text[i];
100:             else
101:                 break;
102:         }
103:         return result;
104:     }
105:
106:     /**
107:      * Returns a string with the MSG name and data
108:      */
109:     public String toString(){
110:         return "MAVLINK_MSG_ID_STATUSTEXT -"+" severity:"+severity+" text:"+text+"
";
111:     }
112: }

```



```

1: // MESSAGE SYSTEM_TIME PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * The system time is the time of the master clock, typically the computer clock of
the main onboard computer.
11:  */
12: public class msg_system_time extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SYSTEM_TIME = 2;
15:     public static final int MAVLINK_MSG_LENGTH = 12;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SYSTEM_TIME;
17:
18:
19:     /**
20:      * Timestamp of the master clock in microseconds since UNIX epoch.
21:      */
22:     public long time_unix_usec;
23:     /**
24:      * Timestamp of the component clock since boot time in milliseconds.
25:      */
26:     public int time_boot_ms;
27:
28:     /**
29:      * Generates the payload for a mavlink message for a message of this type
30:      * @return
31:      */
32:     public MAVLinkPacket pack(){
33:         MAVLinkPacket packet = new MAVLinkPacket();
34:         packet.len = MAVLINK_MSG_LENGTH;
35:         packet.sysid = 255;
36:         packet.compid = 190;
37:         packet.msgid = MAVLINK_MSG_ID_SYSTEM_TIME;
38:         packet.payload.putLong(time_unix_usec);
39:         packet.payload.putInt(time_boot_ms);
40:         return packet;
41:     }
42:
43:     /**
44:      * Decode a system_time message into this class fields
45:      *
46:      * @param payload The message to decode
47:      */
48:     public void unpack(MAVLinkPayload payload) {
49:         payload.resetIndex();
50:         time_unix_usec = payload.getLong();
51:         time_boot_ms = payload.getInt();
52:     }
53:
54:     /**
55:      * Constructor for a new message, just initializes the msgid
56:      */
57:     public msg_system_time(){
58:         msgid = MAVLINK_MSG_ID_SYSTEM_TIME;
59:     }
60:
61:     /**
62:      * Constructor for a new message, initializes the message with the payload
63:      * from a mavlink packet
64:      *
65:      */
66:     public msg_system_time(MAVLinkPacket mavLinkPacket){

```

```

67:         this.sysid = mavLinkPacket.sysid;
68:         this.compid = mavLinkPacket.compid;
69:         this.msgid = MAVLINK_MSG_ID_SYSTEM_TIME;
70:         unpack(mavLinkPacket.payload);
71:         //Log.d("MAVLink", "SYSTEM_TIME");
72:         //Log.d("MAVLINK_MSG_ID_SYSTEM_TIME", toString());
73:     }
74:
75:
76:     /**
77:      * Returns a string with the MSG name and data
78:      */
79:     public String toString(){
80:         return "MAVLINK_MSG_ID_SYSTEM_TIME -"+" time_unix_usec:"+time_unix_usec+"
time_boot_ms:"+time_boot_ms+"";
81:     }
82: }

```



```

1: // MESSAGE SYS_STATUS PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: * The general system state. If the system is following the MAVLink standard, the s
ystem 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 (s
ystem with autonomous position control, position setpoint controlled manually) or AUTO (s
ystem guided by path/waypoint planner). The NAV_MODE defined the current flight state: LI
FTOFF (often an open-loop maneuver), LANDING, WAYPOINTS or VECTOR. This represents the in
ternal navigation state machine. The system status shows wether the system is currently a
ctive or not and if an emergency occurred. During the CRITICAL and EMERGENCY states the MA
V is still considered to be active, but should start emergency procedures autonomously. A
fter a failure occurred it should first move from active to critical to allow manual inter
vention and then move to emergency after a certain timeout.
11: */
12: public class msg_sys_status extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_SYS_STATUS = 1;
15:     public static final int MAVLINK_MSG_LENGTH = 31;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_SYS_STATUS;
17:
18:
19:     /**
20:     * 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: ab
solute pressure, 4: differential pressure, 5: GPS, 6: optical flow, 7: computer vision po
sition, 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
21:     */
22:     public int onboard_control_sensors_present;
23:     /**
24:     * Bitmask showing which onboard controllers and sensors are enabled: Valu
e of 0: not enabled. Value of 1: enabled. Indices: 0: 3D gyro, 1: 3D acc, 2: 3D mag, 3: a
bsolute pressure, 4: differential pressure, 5: GPS, 6: optical flow, 7: computer vision p
osition, 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
25:     */
26:     public int onboard_control_sensors_enabled;
27:     /**
28:     * 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 pos
ition, 13: z/altitude control, 14: x/y position control, 15: motor outputs / control
29:     */
30:     public int onboard_control_sensors_health;
31:     /**
32:     * Maximum usage in percent of the mainloop time, (0%: 0, 100%: 1000) shoul
d be always below 1000
33:     */
34:     public short load;
35:     /**
36:     * Battery voltage, in millivolts (1 = 1 millivolt)
37:     */
38:     public short voltage_battery;
39:     /**
40:     * Battery current, in 10*milliamperes (1 = 10 milliampere), -1: autopilot
does not measure the current

```

```

41:     */
42:     public short current_battery;
43:     /**
44:     * 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)
45:     */
46:     public short drop_rate_comm;
47:     /**
48:     * Communication errors (UART, I2C, SPI, CAN), dropped packets on all links
(packets that were corrupted on reception on the MAV)
49:     */
50:     public short errors_comm;
51:     /**
52:     * Autopilot-specific errors
53:     */
54:     public short errors_count1;
55:     /**
56:     * Autopilot-specific errors
57:     */
58:     public short errors_count2;
59:     /**
60:     * Autopilot-specific errors
61:     */
62:     public short errors_count3;
63:     /**
64:     * Autopilot-specific errors
65:     */
66:     public short errors_count4;
67:     /**
68:     * Remaining battery energy: (0%: 0, 100%: 100), -1: autopilot estimate the
remaining battery
69:     */
70:     public byte battery_remaining;
71:
72:     /**
73:     * Generates the payload for a mavlink message for a message of this type
74:     * @return
75:     */
76:     public MAVLinkPacket pack(){
77:         MAVLinkPacket packet = new MAVLinkPacket();
78:         packet.len = MAVLINK_MSG_LENGTH;
79:         packet.sysid = 255;
80:         packet.compid = 190;
81:         packet.msgid = MAVLINK_MSG_ID_SYS_STATUS;
82:         packet.payload.putInt(onboard_control_sensors_present);
83:         packet.payload.putInt(onboard_control_sensors_enabled);
84:         packet.payload.putInt(onboard_control_sensors_health);
85:         packet.payload.putShort(load);
86:         packet.payload.putShort(voltage_battery);
87:         packet.payload.putShort(current_battery);
88:         packet.payload.putShort(drop_rate_comm);
89:         packet.payload.putShort(errors_comm);
90:         packet.payload.putShort(errors_count1);
91:         packet.payload.putShort(errors_count2);
92:         packet.payload.putShort(errors_count3);
93:         packet.payload.putShort(errors_count4);
94:         packet.payload.putByte(battery_remaining);
95:         return packet;
96:     }
97:
98:     /**
99:     * Decode a sys_status message into this class fields
100:     *
101:     * @param payload The message to decode
102:     */
103:     public void unpack(MAVLinkPayload payload) {
104:         payload.resetIndex();

```

```
105:         onboard_control_sensors_present = payload.getInt();
106:         onboard_control_sensors_enabled = payload.getInt();
107:         onboard_control_sensors_health = payload.getInt();
108:         load = payload.getShort();
109:         voltage_battery = payload.getShort();
110:         current_battery = payload.getShort();
111:         drop_rate_comm = payload.getShort();
112:         errors_comm = payload.getShort();
113:         errors_count1 = payload.getShort();
114:         errors_count2 = payload.getShort();
115:         errors_count3 = payload.getShort();
116:         errors_count4 = payload.getShort();
117:         battery_remaining = payload.getByte();
118:     }
119:
120:     /**
121:      * Constructor for a new message, just initializes the msgid
122:      */
123:     public msg_sys_status(){
124:         msgid = MAVLINK_MSG_ID_SYS_STATUS;
125:     }
126:
127:     /**
128:      * Constructor for a new message, initializes the message with the payload
129:      * from a mavlink packet
130:      */
131:     public msg_sys_status(MAVLinkPacket mavLinkPacket){
132:         this.sysid = mavLinkPacket.sysid;
133:         this.compid = mavLinkPacket.compid;
134:         this.msgid = MAVLINK_MSG_ID_SYS_STATUS;
135:         unpack(mavLinkPacket.payload);
136:         //Log.d("MAVLink", "SYS_STATUS");
137:         //Log.d("MAVLINK_MSG_ID_SYS_STATUS", toString());
138:     }
139:
140:
141:
142:     /**
143:      * Returns a string with the MSG name and data
144:      */
145:     public String toString(){
146:         return "MAVLINK_MSG_ID_SYS_STATUS -"+" onboard_control_sensors_present:"+onboard_control_sensors_present+" onboard_control_sensors_enabled:"+onboard_control_sensors_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_count2:"+errors_count2+" errors_count3:"+errors_count3+" errors_count4:"+errors_count4+" battery_remaining:"+battery_remaining+"";
147:     }
148: }
```

```

1: // MESSAGE VFR_HUD PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Metrics typically displayed on a HUD for fixed wing aircraft
11:  */
12: public class msg_vfr_hud extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_VFR_HUD = 74;
15:     public static final int MAVLINK_MSG_LENGTH = 20;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_VFR_HUD;
17:
18:
19:     /**
20:      * Current airspeed in m/s
21:      */
22:     public float airspeed;
23:     /**
24:      * Current ground speed in m/s
25:      */
26:     public float groundspeed;
27:     /**
28:      * Current altitude (MSL), in meters
29:      */
30:     public float alt;
31:     /**
32:      * Current climb rate in meters/second
33:      */
34:     public float climb;
35:     /**
36:      * Current heading in degrees, in compass units (0..360, 0=north)
37:      */
38:     public short heading;
39:     /**
40:      * Current throttle setting in integer percent, 0 to 100
41:      */
42:     public short throttle;
43:
44:     /**
45:      * Generates the payload for a mavlink message for a message of this type
46:      * @return
47:      */
48:     public MAVLinkPacket pack(){
49:         MAVLinkPacket packet = new MAVLinkPacket();
50:         packet.len = MAVLINK_MSG_LENGTH;
51:         packet.sysid = 255;
52:         packet.compid = 190;
53:         packet.msgid = MAVLINK_MSG_ID_VFR_HUD;
54:         packet.payload.putFloat(airspeed);
55:         packet.payload.putFloat(groundspeed);
56:         packet.payload.putFloat(alt);
57:         packet.payload.putFloat(climb);
58:         packet.payload.putShort(heading);
59:         packet.payload.putShort(throttle);
60:         return packet;
61:     }
62:
63:     /**
64:      * Decode a vfr_hud message into this class fields
65:      *
66:      * @param payload The message to decode
67:      */

```

```

68:     public void unpack(MAVLinkPayload payload) {
69:         payload.resetIndex();
70:         airspeed = payload.getFloat();
71:         groundspeed = payload.getFloat();
72:         alt = payload.getFloat();
73:         climb = payload.getFloat();
74:         heading = payload.getShort();
75:         throttle = payload.getShort();
76:     }
77:
78:     /**
79:      * Constructor for a new message, just initializes the msgid
80:      */
81:     public msg_vfr_hud(){
82:         msgid = MAVLINK_MSG_ID_VFR_HUD;
83:     }
84:
85:     /**
86:      * Constructor for a new message, initializes the message with the payload
87:      * from a mavlink packet
88:      */
89:     public msg_vfr_hud(MAVLinkPacket mavLinkPacket){
90:         this.sysid = mavLinkPacket.sysid;
91:         this.compid = mavLinkPacket.compid;
92:         this.msgid = MAVLINK_MSG_ID_VFR_HUD;
93:         unpack(mavLinkPacket.payload);
94:         //Log.d("MAVLink", "VFR_HUD");
95:         //Log.d("MAVLINK_MSG_ID_VFR_HUD", toString());
96:     }
97:
98:
99:
100:     /**
101:      * Returns a string with the MSG name and data
102:      */
103:     public String toString(){
104:         return "MAVLINK_MSG_ID_VFR_HUD -"+" airspeed:"+airspeed+" groundspeed:"+gr
105:         oundspeed+" alt:"+alt+" climb:"+climb+" heading:"+heading+" throttle:"+throttle+"";
106:     }

```



```

1: // MESSAGE VICON_POSITION_ESTIMATE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: *
11: */
12: public class msg_vicon_position_estimate extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE = 104;
15:     public static final int MAVLINK_MSG_LENGTH = 32;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_VICON_POSITION
_ESTIMATE;
17:
18:
19:     /**
20:     * Timestamp (microseconds, synced to UNIX time or since system boot)
21:     */
22:     public long usec;
23:     /**
24:     * Global X position
25:     */
26:     public float x;
27:     /**
28:     * Global Y position
29:     */
30:     public float y;
31:     /**
32:     * Global Z position
33:     */
34:     public float z;
35:     /**
36:     * Roll angle in rad
37:     */
38:     public float roll;
39:     /**
40:     * Pitch angle in rad
41:     */
42:     public float pitch;
43:     /**
44:     * Yaw angle in rad
45:     */
46:     public float yaw;
47:
48:     /**
49:     * Generates the payload for a mavlink message for a message of this type
50:     * @return
51:     */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE;
58:         packet.payload.putLong(usec);
59:         packet.payload.putFloat(x);
60:         packet.payload.putFloat(y);
61:         packet.payload.putFloat(z);
62:         packet.payload.putFloat(roll);
63:         packet.payload.putFloat(pitch);
64:         packet.payload.putFloat(yaw);
65:         return packet;
66:     }
67:
68:
69:     /**
70:     * Decode a vicon_position_estimate message into this class fields
71:     * @param payload The message to decode
72:     */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         usec = payload.getLong();
76:         x = payload.getFloat();
77:         y = payload.getFloat();
78:         z = payload.getFloat();
79:         roll = payload.getFloat();
80:         pitch = payload.getFloat();
81:         yaw = payload.getFloat();
82:     }
83:
84:     /**
85:     * Constructor for a new message, just initializes the msgid
86:     */
87:     public msg_vicon_position_estimate(){
88:         msgid = MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE;
89:     }
90:
91:     /**
92:     * Constructor for a new message, initializes the message with the payload
93:     * from a mavlink packet
94:     */
95:     public msg_vicon_position_estimate(MAVLinkPacket mavLinkPacket){
96:         this.sysid = mavLinkPacket.sysid;
97:         this.compid = mavLinkPacket.compid;
98:         this.msgid = MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE;
99:         unpack(mavLinkPacket.payload);
100:         //Log.d("MAVLink", "VICON_POSITION_ESTIMATE");
101:         //Log.d("MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE", toString());
102:     }
103:
104:
105:     /**
106:     * Returns a string with the MSG name and data
107:     */
108:     public String toString(){
109:         return "MAVLINK_MSG_ID_VICON_POSITION_ESTIMATE -"+" usec:"+usec+" x:"+x+"
y:"+y+" z:"+z+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+"";
110:     }
111:
112: }

```



```

1: // MESSAGE VISION_POSITION_ESTIMATE PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10: *
11: */
12: public class msg_vision_position_estimate extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE = 102;
15:     public static final int MAVLINK_MSG_LENGTH = 32;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_VISION_POSITIO
N_ESTIMATE;
17:
18:
19:     /**
20:     * Timestamp (microseconds, synced to UNIX time or since system boot)
21:     */
22:     public long usec;
23:     /**
24:     * Global X position
25:     */
26:     public float x;
27:     /**
28:     * Global Y position
29:     */
30:     public float y;
31:     /**
32:     * Global Z position
33:     */
34:     public float z;
35:     /**
36:     * Roll angle in rad
37:     */
38:     public float roll;
39:     /**
40:     * Pitch angle in rad
41:     */
42:     public float pitch;
43:     /**
44:     * Yaw angle in rad
45:     */
46:     public float yaw;
47:
48:     /**
49:     * Generates the payload for a mavlink message for a message of this type
50:     * @return
51:     */
52:     public MAVLinkPacket pack(){
53:         MAVLinkPacket packet = new MAVLinkPacket();
54:         packet.len = MAVLINK_MSG_LENGTH;
55:         packet.sysid = 255;
56:         packet.compid = 190;
57:         packet.msgid = MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE;
58:         packet.payload.putLong(usec);
59:         packet.payload.putFloat(x);
60:         packet.payload.putFloat(y);
61:         packet.payload.putFloat(z);
62:         packet.payload.putFloat(roll);
63:         packet.payload.putFloat(pitch);
64:         packet.payload.putFloat(yaw);
65:         return packet;
66:     }
67:
68:     /**
69:     * Decode a vision_position_estimate message into this class fields
70:     *
71:     * @param payload The message to decode
72:     */
73:     public void unpack(MAVLinkPayload payload) {
74:         payload.resetIndex();
75:         usec = payload.getLong();
76:         x = payload.getFloat();
77:         y = payload.getFloat();
78:         z = payload.getFloat();
79:         roll = payload.getFloat();
80:         pitch = payload.getFloat();
81:         yaw = payload.getFloat();
82:     }
83:
84:     /**
85:     * Constructor for a new message, just initializes the msgid
86:     */
87:     public msg_vision_position_estimate(){
88:         msgid = MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE;
89:     }
90:
91:     /**
92:     * Constructor for a new message, initializes the message with the payload
93:     * from a mavlink packet
94:     */
95:     public msg_vision_position_estimate(MAVLinkPacket mavLinkPacket){
96:         this.sysid = mavLinkPacket.sysid;
97:         this.compid = mavLinkPacket.compid;
98:         this.msgid = MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE;
99:         unpack(mavLinkPacket.payload);
100:         //Log.d("MAVLink", "VISION_POSITION_ESTIMATE");
101:         //Log.d("MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE", toString());
102:     }
103:
104:
105:     /**
106:     * Returns a string with the MSG name and data
107:     */
108:     public String toString(){
109:         return "MAVLINK_MSG_ID_VISION_POSITION_ESTIMATE -"+" usec:"+usec+" x:"+x+"
y:"+y+" z:"+z+" roll:"+roll+" pitch:"+pitch+" yaw:"+yaw+"";
110:     }
111: }
112: }

```



```

1:  // MESSAGE VISION_SPEED_ESTIMATE PACKING
2:  package com.MAVLink.Messages.ardupilotmega;
3:
4:  import com.MAVLink.Messages.MAVLinkMessage;
5:  import com.MAVLink.Messages.MAVLinkPayload;
6:  import com.MAVLink.Messages.MAVLinkPacket;
7:  //import android.util.Log;
8:
9:  /**
10:   *
11:   */
12:  public class msg_vision_speed_estimate extends MAVLinkMessage{
13:
14:      public static final int MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE = 103;
15:      public static final int MAVLINK_MSG_LENGTH = 20;
16:      private static final long serialVersionUID = MAVLINK_MSG_ID_VISION_SPEED_E
STIMATE;
17:
18:
19:      /**
20:       * Timestamp (microseconds, synced to UNIX time or since system boot)
21:       */
22:      public long usec;
23:      /**
24:       * Global X speed
25:       */
26:      public float x;
27:      /**
28:       * Global Y speed
29:       */
30:      public float y;
31:      /**
32:       * Global Z speed
33:       */
34:      public float z;
35:
36:      /**
37:       * Generates the payload for a mavlink message for a message of this type
38:       * @return
39:       */
40:      public MAVLinkPacket pack(){
41:          MAVLinkPacket packet = new MAVLinkPacket();
42:          packet.len = MAVLINK_MSG_LENGTH;
43:          packet.sysid = 255;
44:          packet.compuid = 190;
45:          packet.msgid = MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE;
46:          packet.payload.putLong(usec);
47:          packet.payload.putFloat(x);
48:          packet.payload.putFloat(y);
49:          packet.payload.putFloat(z);
50:          return packet;
51:      }
52:
53:      /**
54:       * Decode a vision_speed_estimate message into this class fields
55:       *
56:       * @param payload The message to decode
57:       */
58:      public void unpack(MAVLinkPayload payload) {
59:          payload.resetIndex();
60:          usec = payload.getLong();
61:          x = payload.getFloat();
62:          y = payload.getFloat();
63:          z = payload.getFloat();
64:      }
65:
66:      /**

```

```

67:      * Constructor for a new message, just initializes the msgid
68:      */
69:      public msg_vision_speed_estimate(){
70:          msgid = MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE;
71:      }
72:
73:      /**
74:       * Constructor for a new message, initializes the message with the payload
75:       * from a mavlink packet
76:       */
77:      public msg_vision_speed_estimate(MAVLinkPacket mavLinkPacket){
78:          this.sysid = mavLinkPacket.sysid;
79:          this.compuid = mavLinkPacket.compuid;
80:          this.msgid = MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE;
81:          unpack(mavLinkPacket.payload);
82:          //Log.d("MAVLink", "VISION_SPEED_ESTIMATE");
83:          //Log.d("MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE", toString());
84:      }
85:
86:
87:
88:      /**
89:       * Returns a string with the MSG name and data
90:       */
91:      public String toString(){
92:          return "MAVLINK_MSG_ID_VISION_SPEED_ESTIMATE -"+" usec:"+usec+" x:"+x+" y:
"+y+" z:"+z+"";
93:      }
94:  }

```



```

1: // MESSAGE WIND PACKING
2: package com.MAVLink.Messages.ardupilotmega;
3:
4: import com.MAVLink.Messages.MAVLinkMessage;
5: import com.MAVLink.Messages.MAVLinkPayload;
6: import com.MAVLink.Messages.MAVLinkPacket;
7: //import android.util.Log;
8:
9: /**
10:  * Wind estimation
11:  */
12: public class msg_wind extends MAVLinkMessage{
13:
14:     public static final int MAVLINK_MSG_ID_WIND = 168;
15:     public static final int MAVLINK_MSG_LENGTH = 12;
16:     private static final long serialVersionUID = MAVLINK_MSG_ID_WIND;
17:
18:
19:     /**
20:      * wind direction that wind is coming from (degrees)
21:      */
22:     public float direction;
23:     /**
24:      * wind speed in ground plane (m/s)
25:      */
26:     public float speed;
27:     /**
28:      * vertical wind speed (m/s)
29:      */
30:     public float speed_z;
31:
32:     /**
33:      * Generates the payload for a mavlink message for a message of this type
34:      * @return
35:      */
36:     public MAVLinkPacket pack(){
37:         MAVLinkPacket packet = new MAVLinkPacket();
38:         packet.len = MAVLINK_MSG_LENGTH;
39:         packet.sysid = 255;
40:         packet.compid = 190;
41:         packet.msgid = MAVLINK_MSG_ID_WIND;
42:         packet.payload.putFloat(direction);
43:         packet.payload.putFloat(speed);
44:         packet.payload.putFloat(speed_z);
45:         return packet;
46:     }
47:
48:     /**
49:      * Decode a wind message into this class fields
50:      *
51:      * @param payload The message to decode
52:      */
53:     public void unpack(MAVLinkPayload payload) {
54:         payload.resetIndex();
55:         direction = payload.getFloat();
56:         speed = payload.getFloat();
57:         speed_z = payload.getFloat();
58:     }
59:
60:     /**
61:      * Constructor for a new message, just initializes the msgid
62:      */
63:     public msg_wind(){
64:         msgid = MAVLINK_MSG_ID_WIND;
65:     }
66:
67:     /**

```

```

68:      * Constructor for a new message, initializes the message with the payload
69:      * from a mavlink packet
70:      */
71:     */
72:     public msg_wind(MAVLinkPacket mavLinkPacket){
73:         this.sysid = mavLinkPacket.sysid;
74:         this.compid = mavLinkPacket.compid;
75:         this.msgid = MAVLINK_MSG_ID_WIND;
76:         unpack(mavLinkPacket.payload);
77:         //Log.d("MAVLink", "WIND");
78:         //Log.d("MAVLINK_MSG_ID_WIND", toString());
79:     }
80:
81:
82:     /**
83:      * Returns a string with the MSG name and data
84:      */
85:     public String toString(){
86:         return "MAVLINK_MSG_ID_WIND -"+" direction:"+direction+" speed:"+speed+" s
87:         peed_z:"+speed_z+"";
88:     }

```



```

1:  package com.MAVLink.Messages;
2:
3:  /**
4:   * X.25 CRC calculation for MAVlink messages. The checksum must be initialized,
5:   * updated with witch field of the message, and then finished with the message
6:   * id.
7:   *
8:   */
9:  public class CRC {
10:     private final int[] MAVLINK_MESSAGE_CRCS = {50, 124, 137, 0, 237, 217, 104,
11:     , 119, 0, 0, 0, 89, 0, 0, 0, 0, 0, 0, 0, 0, 0, 214, 159, 220, 168, 24, 23, 170, 144, 67, 115
12:     , 39, 246, 185, 104, 237, 244, 222, 212, 9, 254, 230, 28, 28, 132, 221, 232, 11, 153, 41,
13:     , 39, 214, 223, 141, 33, 15, 3, 100, 24, 239, 238, 30, 240, 183, 130, 130, 0, 148, 21, 0,
14:     , 243, 124, 0, 0, 0, 20, 0, 152, 143, 0, 0, 127, 106, 0, 0, 0, 0, 0, 0, 231, 183, 63, 54
15:     , 0, 0, 0, 0, 0, 0, 0, 175, 102, 158, 208, 56, 93, 0, 0, 0, 0, 235, 93, 124, 0, 0, 0, 0,
16:     , 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,
17:     , 42, 241, 15, 134, 219, 208, 188, 84, 22, 19, 21, 134, 0, 78, 68, 189, 127, 111, 21, 21,
18:     , 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,
19:     , 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,
20:     , 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,
21:     , 44, 83, 46, 0};
22:
23:     private static final int CRC_INIT_VALUE = 0xffff;
24:     private int CRCvalue;
25:
26:     /**
27:      * Accumulate the X.25 CRC by adding one char at a time.
28:      *
29:      * The checksum function adds the hash of one char at a time to the 16 bit
30:      * checksum (uint16_t).
31:      *
32:      * @param data
33:      *      new char to hash
34:      * @param crcAccum
35:      *      the already accumulated checksum
36:      */
37:     public void update_checksum(int data) {
38:         int tmp;
39:         data= data & 0xff;          //cast because we want an unsigned type
40:         tmp = data ^ (CRCvalue & 0xff);
41:         tmp ^= (tmp << 4) & 0xff;
42:         CRCvalue = ((CRCvalue >> 8) & 0xff) ^ (tmp << 8) ^ (tmp << 3)
43:             ^ ((tmp >> 4) & 0xf);
44:     }
45:
46:     /**
47:      * Finish the CRC calculation of a message, by running the CRC with the
48:      * Magic Byte. This Magic byte has been defined in MAVlink v1.0.
49:      *
50:      * @param msgid
51:      *      The message id number
52:      */
53:     public void finish_checksum(int msgid) {
54:         update_checksum(MAVLINK_MESSAGE_CRCS[msgid]);
55:     }
56:
57:     /**
58:      * Initialize the buffer for the X.25 CRC
59:      *
60:      */
61:     public void start_checksum() {
62:         CRCvalue = CRC_INIT_VALUE;
63:     }
64:
65:     public int getMSB() {
66:         return ((CRCvalue >> 8) & 0xff);
67:     }
68:
69: }

```

```

57:     public int getLSB() {
58:         return (CRCvalue & 0xff);
59:     }
60:
61:     public CRC() {
62:         start_checksum();
63:     }
64:
65: }

```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class FENCE_ACTION {
6:     public static final int FENCE_ACTION_NONE = 0; /* Disable fenced mode | */
7:     public static final int FENCE_ACTION_GUIDED = 1; /* Switched to guided mode to return point (fence point 0) | */
8:     public static final int FENCE_ACTION_REPORT = 2; /* Report fence breach, but don't take action | */
9:     public static final int FENCE_ACTION_ENUM_END = 3; /* | */
10: }
```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class FENCE_BREACH {
6:     public static final int FENCE_BREACH_NONE = 0; /* No last fence breach | *
/
7:     public static final int FENCE_BREACH_MINALT = 1; /* Breached minimum altit
ude | */
8:     public static final int FENCE_BREACH_MAXALT = 2; /* Breached minimum altit
ude | */
9:     public static final int FENCE_BREACH_BOUNDARY = 3; /* Breached fence bound
ary | */
10:    public static final int FENCE_BREACH_ENUM_END = 4; /* | */
11: }
```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class LIMITS_STATE {
6:     public static final int LIMITS_INIT = 0; /* pre-initialization | */
7:     public static final int LIMITS_DISABLED = 1; /* disabled | */
8:     public static final int LIMITS_ENABLED = 2; /* checking limits | */
9:     public static final int LIMITS_TRIGGERED = 3; /* a limit has been breache
d | */
10:     public static final int LIMITS_RECOVERING = 4; /* taking action eg. RTL |
*/
11:     public static final int LIMITS_RECOVERED = 5; /* we're no longer in breac
h of a limit | */
12:     public static final int LIMITS_STATE_ENUM_END = 6; /* | */
13: }
```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class LIMIT_MODULE {
6:     public static final int LIMIT_GPSLOCK = 1; /* pre-initialization | */
7:     public static final int LIMIT_GEOFENCE = 2; /* disabled | */
8:     public static final int LIMIT_ALTITUDE = 4; /* checking limits | */
9:     public static final int LIMIT_MODULE_ENUM_END = 5; /* | */
10: }
```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAVLINK_DATA_STREAM_TYPE {
6:     public static final int MAVLINK_DATA_STREAM_IMG_JPEG = 1; /* | */
7:     public static final int MAVLINK_DATA_STREAM_IMG_BMP = 2; /* | */
8:     public static final int MAVLINK_DATA_STREAM_IMG_RAW8U = 3; /* | */
9:     public static final int MAVLINK_DATA_STREAM_IMG_RAW32U = 4; /* | */
10:    public static final int MAVLINK_DATA_STREAM_IMG_PGM = 5; /* | */
11:    public static final int MAVLINK_DATA_STREAM_IMG_PNG = 6; /* | */
12:    public static final int MAVLINK_DATA_STREAM_TYPE_ENUM_END = 7; /* | */
13: }
```



```
1: /** Micro air vehicle / autopilot classes. This identifies the individual model.
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_AUTOPILOT {
6:     public static final int MAV_AUTOPILOT_GENERIC = 0; /* Generic autopilot, f
ull support for everything | */
7:     public static final int MAV_AUTOPILOT_PIXHAWK = 1; /* PIXHAWK autopilot, h
ttp://pixhawk.ethz.ch | */
8:     public static final int MAV_AUTOPILOT_SLUGS = 2; /* SLUGS autopilot, http:
//slugsuav.soe.ucsc.edu | */
9:     public static final int MAV_AUTOPILOT_ARDUPILOTMEGA = 3; /* ArduPilotMega
/ ArduCopter, http://diydrones.com | */
10:    public static final int MAV_AUTOPILOT_OPENPILOT = 4; /* OpenPilot, http://
openpilot.org | */
11:    public static final int MAV_AUTOPILOT_GENERIC_WAYPOINTS_ONLY = 5; /* Gener
ic autopilot only supporting simple waypoints | */
12:    public static final int MAV_AUTOPILOT_GENERIC_WAYPOINTS_AND_SIMPLE_NAVIGAT
ION_ONLY = 6; /* Generic autopilot supporting waypoints and other simple naviga
tion commands | */
13:    public static final int MAV_AUTOPILOT_GENERIC_MISSION_FULL = 7; /* Generic
autopilot supporting the full mission command set | */
14:    public static final int MAV_AUTOPILOT_INVALID = 8; /* No valid autopilot,
e.g. a GCS or other MAVLink component | */
15:    public static final int MAV_AUTOPILOT_PPZ = 9; /* PPZ UAV - http://nongnu.
org/paparazzi | */
16:    public static final int MAV_AUTOPILOT_UDB = 10; /* UAV Dev Board | */
17:    public static final int MAV_AUTOPILOT_FP = 11; /* FlexiPilot | */
18:    public static final int MAV_AUTOPILOT_PX4 = 12; /* PX4 Autopilot - http://
pixhawk.ethz.ch/px4/ | */
19:    public static final int MAV_AUTOPILOT_ENUM_END = 13; /* | */
20: }
```



```

1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_CMD {
6:     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| */
7:     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| */
8:     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| */
9:     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| */
10:    public static final int MAV_CMD_NAV_RETURN_TO_LAUNCH = 20; /* Return to launch location|Empty| Empty| Empty| Empty| Empty| Empty| Empty| */
11:    public static final int MAV_CMD_NAV_LAND = 21; /* Land at location|Empty| Empty| Empty| Desired yaw angle.| Latitude| Longitude| Altitude| */
12:    public static final int MAV_CMD_NAV_TAKEOFF = 22; /* Takeoff from ground|Empty| Minimum pitch (if airspeed sensor present), desired pitch without sensor| Empty| Empty| Yaw angle (if magnetometer present), ignored without magnetometer| Latitude| Longitude| Altitude| */
13:    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 interest 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| */
14:    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| */
15:    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| */
16:    public static final int MAV_CMD_CONDITION_DELAY = 112; /* Delay mission state machine. |Delay in seconds (decimal)| Empty| Empty| Empty| Empty| Empty| Empty| */
17:    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| */
18:    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| */
19:    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| */
20:    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| */
21:    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| */
22:    public static final int MAV_CMD_DO_JUMP = 177; /* Jump to the desired command and in the mission list. Repeat this action only the specified number of times|Sequence

```

```

number| Repeat count| Empty| Empty| Empty| Empty| Empty| */
23:    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| */
24:    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| */
25:    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| */
26:    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| */
27:    public static final int MAV_CMD_DO_REPEAT_RELAY = 182; /* Cycle a relay on and off for a desired number of cycles with a desired period. |Relay number| Cycle count| Cycle time (seconds, decimal)| Empty| Empty| Empty| Empty| */
28:    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| */
29:    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, decimal)| Empty| Empty| Empty| */
30:    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| */
31:    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)| */
32:    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| */
33:    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| */
34:    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| */
35:    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| */
36:    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| */
37:    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| */
38:    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| */
39:      public static final int MAV_CMD_PREFLIGHT_REBOOT_SHUTDOWN = 246; /* Request the reboot or shutdown of system components. |0: Do nothing for autopilot, 1: Reboot autopilot, 2: Shutdown autopilot. | 0: Do nothing for onboard computer, 1: Reboot onboard computer, 2: Shutdown onboard computer. | Reserved| Reserved| Empty| Empty| Empty| */
40:      public static final int MAV_CMD_OVERRIDE_GOTO = 252; /* Hold / continue the 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 point| Desired yaw angle in degrees| Latitude / X position| Longitude / Y position| Altitude / Z position| */
41:      public static final int MAV_CMD_MISSION_START = 300; /* start running a mission |first_item: the first mission item to run| last_item: the last mission item to run (after this item is run, the mission ends)| */
42:      public static final int MAV_CMD_COMPONENT_ARM_DISARM = 400; /* Arms / Disarms a component |1 to arm, 0 to disarm| */
43:      public static final int MAV_CMD_ENUM_END = 401; /* | */
44: }

```



```

1: /** ACK / NACK / ERROR values as a result of MAV_CMDs and for mission item transmi
ssion.
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_CMD_ACK {
6:     public static final int MAV_CMD_ACK_OK = 1; /* Command / mission item is o
k. | */
7:     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. | */
/
8:     public static final int MAV_CMD_ACK_ERR_ACCESS_DENIED = 3; /* The system i
s refusing to accept this command from this source / communication partner. | */
9:     public static final int MAV_CMD_ACK_ERR_NOT_SUPPORTED = 4; /* Command or m
ission item is not supported, other commands would be accepted. | */
10:    public static final int MAV_CMD_ACK_ERR_COORDINATE_FRAME_NOT_SUPPORTED = 5
; /* The coordinate frame of this command / mission item is not supported. | */
11:    public static final int MAV_CMD_ACK_ERR_COORDINATES_OUT_OF_RANGE = 6; /* T
he coordinate frame of this command is ok, but he coordinate values exceed the safety lim
its of this system. This is a generic error, please use the more specific error messages
below if possible. | */
12:    public static final int MAV_CMD_ACK_ERR_X_LAT_OUT_OF_RANGE = 7; /* The X o
r latitude value is out of range. | */
13:    public static final int MAV_CMD_ACK_ERR_Y_LON_OUT_OF_RANGE = 8; /* The Y o
r longitude value is out of range. | */
14:    public static final int MAV_CMD_ACK_ERR_Z_ALT_OUT_OF_RANGE = 9; /* The Z o
r altitude value is out of range. | */
15:    public static final int MAV_CMD_ACK_ENUM_END = 10; /* | */
16: }

```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_COMPONENT {
6:     public static final int MAV_COMP_ID_ALL = 0; /* | */
7:     public static final int MAV_COMP_ID_CAMERA = 100; /* | */
8:     public static final int MAV_COMP_ID_SERVO1 = 140; /* | */
9:     public static final int MAV_COMP_ID_SERVO2 = 141; /* | */
10:    public static final int MAV_COMP_ID_SERVO3 = 142; /* | */
11:    public static final int MAV_COMP_ID_SERVO4 = 143; /* | */
12:    public static final int MAV_COMP_ID_SERVO5 = 144; /* | */
13:    public static final int MAV_COMP_ID_SERVO6 = 145; /* | */
14:    public static final int MAV_COMP_ID_SERVO7 = 146; /* | */
15:    public static final int MAV_COMP_ID_SERVO8 = 147; /* | */
16:    public static final int MAV_COMP_ID_SERVO9 = 148; /* | */
17:    public static final int MAV_COMP_ID_SERVO10 = 149; /* | */
18:    public static final int MAV_COMP_ID_SERVO11 = 150; /* | */
19:    public static final int MAV_COMP_ID_SERVO12 = 151; /* | */
20:    public static final int MAV_COMP_ID_SERVO13 = 152; /* | */
21:    public static final int MAV_COMP_ID_SERVO14 = 153; /* | */
22:    public static final int MAV_COMP_ID_MAPPER = 180; /* | */
23:    public static final int MAV_COMP_ID_MISSIONPLANNER = 190; /* | */
24:    public static final int MAV_COMP_ID_PATHPLANNER = 195; /* | */
25:    public static final int MAV_COMP_ID_IMU = 200; /* | */
26:    public static final int MAV_COMP_ID_IMU_2 = 201; /* | */
27:    public static final int MAV_COMP_ID_IMU_3 = 202; /* | */
28:    public static final int MAV_COMP_ID_GPS = 220; /* | */
29:    public static final int MAV_COMP_ID_UDP_BRIDGE = 240; /* | */
30:    public static final int MAV_COMP_ID_UART_BRIDGE = 241; /* | */
31:    public static final int MAV_COMP_ID_SYSTEM_CONTROL = 250; /* | */
32:    public static final int MAV_COMPONENT_ENUM_END = 251; /* | */
33: }
```



```

1:  /** Data stream IDs. A data stream is not a fixed set of messages, but rather a
2:      recommendation to the autopilot software. Individual autopilots may or may not
3:      obey the recommended messages.
4:  */
5:  package com.MAVLink.Messages.enums;
6:
7:  public class MAV_DATA_STREAM {
8:      public static final int MAV_DATA_STREAM_ALL = 0; /* Enable all data stream
9: s | */
10:      public static final int MAV_DATA_STREAM_RAW_SENSORS = 1; /* Enable IMU_RAW
11: , GPS_RAW, GPS_STATUS packets. | */
12:      public static final int MAV_DATA_STREAM_EXTENDED_STATUS = 2; /* Enable GPS
13: _STATUS, CONTROL_STATUS, AUX_STATUS | */
14:      public static final int MAV_DATA_STREAM_RC_CHANNELS = 3; /* Enable RC_CHAN
15: NELS_SCALED, RC_CHANNELS_RAW, SERVO_OUTPUT_RAW | */
16:      public static final int MAV_DATA_STREAM_RAW_CONTROLLER = 4; /* Enable ATTI
17: TUDE_CONTROLLER_OUTPUT, POSITION_CONTROLLER_OUTPUT, NAV_CONTROLLER_OUTPUT. | */
18:      public static final int MAV_DATA_STREAM_POSITION = 6; /* Enable LOCAL_POSI
19: TION, GLOBAL_POSITION/GLOBAL_POSITION_INT messages. | */
20:      public static final int MAV_DATA_STREAM_EXTRA1 = 10; /* Dependent on the a
21: utopilot | */
22:      public static final int MAV_DATA_STREAM_EXTRA2 = 11; /* Dependent on the a
23: utopilot | */
24:      public static final int MAV_DATA_STREAM_EXTRA3 = 12; /* Dependent on the a
25: utopilot | */
26:      public static final int MAV_DATA_STREAM_ENUM_END = 13; /* | */
27:  }

```



```

1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_FRAME {
6:     public static final int MAV_FRAME_GLOBAL = 0; /* Global coordinate frame,
WGS84 coordinate system. First value / x: latitude, second value / y: longitude, third va
lue / z: positive altitude over mean sea level (MSL) | */
7:     public static final int MAV_FRAME_LOCAL_NED = 1; /* Local coordinate frame
, Z-up (x: north, y: east, z: down). | */
8:     public static final int MAV_FRAME_MISSION = 2; /* NOT a coordinate frame,
indicates a mission command. | */
9:     public static final int MAV_FRAME_GLOBAL_RELATIVE_ALT = 3; /* Global coord
inate frame, WGS84 coordinate system, relative altitude over ground with respect to the h
ome position. First value / x: latitude, second value / y: longitude, third value / z: po
sitive altitude with 0 being at the altitude of the home location. | */
10:    public static final int MAV_FRAME_LOCAL_ENU = 4; /* Local coordinate frame
, Z-down (x: east, y: north, z: up) | */
11:    public static final int MAV_FRAME_ENUM_END = 5; /* | */
12: }

```



```

1: /** Override command, pauses current mission execution and moves immediately to a
position
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_GOTO {
6:     public static final int MAV_GOTO_DO_HOLD = 0; /* Hold at the current posit
ion. | */
7:     public static final int MAV_GOTO_DO_CONTINUE = 1; /* Continue with the nex
t item in mission execution. | */
8:     public static final int MAV_GOTO_HOLD_AT_CURRENT_POSITION = 2; /* Hold at
the current position of the system | */
9:     public static final int MAV_GOTO_HOLD_AT_SPECIFIED_POSITION = 3; /* Hold a
t the position specified in the parameters of the DO_HOLD action | */
10:    public static final int MAV_GOTO_ENUM_END = 4; /* | */
11: }

```



```
1: /** result in a mavlink mission ack
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_MISSION_RESULT {
6:     public static final int MAV_MISSION_ACCEPTED = 0; /* mission accepted OK |
*/
7:     public static final int MAV_MISSION_ERROR = 1; /* generic error / not acce
pting mission commands at all right now | */
8:     public static final int MAV_MISSION_UNSUPPORTED_FRAME = 2; /* coordinate f
rame is not supported | */
9:     public static final int MAV_MISSION_UNSUPPORTED = 3; /* command is not sup
ported | */
10:    public static final int MAV_MISSION_NO_SPACE = 4; /* mission item exceeds
storage space | */
11:    public static final int MAV_MISSION_INVALID = 5; /* one of the parameters
has an invalid value | */
12:    public static final int MAV_MISSION_INVALID_PARAM1 = 6; /* param1 has an i
nvalid value | */
13:    public static final int MAV_MISSION_INVALID_PARAM2 = 7; /* param2 has an i
nvalid value | */
14:    public static final int MAV_MISSION_INVALID_PARAM3 = 8; /* param3 has an i
nvalid value | */
15:    public static final int MAV_MISSION_INVALID_PARAM4 = 9; /* param4 has an i
nvalid value | */
16:    public static final int MAV_MISSION_INVALID_PARAM5_X = 10; /* x/param5 has
an invalid value | */
17:    public static final int MAV_MISSION_INVALID_PARAM6_Y = 11; /* y/param6 has
an invalid value | */
18:    public static final int MAV_MISSION_INVALID_PARAM7 = 12; /* param7 has an
invalid value | */
19:    public static final int MAV_MISSION_INVALID_SEQUENCE = 13; /* received way
point out of sequence | */
20:    public static final int MAV_MISSION_DENIED = 14; /* not accepting any miss
ion commands from this communication partner | */
21:    public static final int MAV_MISSION_RESULT_ENUM_END = 15; /* | */
22: }
```



```

1: /** These defines are predefined OR-combined mode flags. There is no need to use v
alues from this enum, but it
2:     simplifies the use of the mode flags. Note that manual input is ena
bled in all modes as a safety override.
3: */
4: package com.MAVLink.Messages.enums;
5:
6: public class MAV_MODE {
7:     public static final int MAV_MODE_PREFLIGHT = 0; /* System is not ready to
fly, booting, calibrating, etc. No flag is set. | */
8:     public static final int MAV_MODE_MANUAL_DISARMED = 64; /* System is allowe
d to be active, under manual (RC) control, no stabilization | */
9:     public static final int MAV_MODE_TEST_DISARMED = 66; /* UNDEFINED mode. Th
is solely depends on the autopilot - use with caution, intended for developers only. | */
10:    public static final int MAV_MODE_STABILIZE_DISARMED = 80; /* System is all
owed to be active, under assisted RC control. | */
11:    public static final int MAV_MODE_GUIDED_DISARMED = 88; /* System is allowe
d to be active, under autonomous control, manual setpoint | */
12:    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) | */
13:    public static final int MAV_MODE_MANUAL_ARMED = 192; /* System is allowed
to be active, under manual (RC) control, no stabilization | */
14:    public static final int MAV_MODE_TEST_ARMED = 194; /* UNDEFINED mode. This
solely depends on the autopilot - use with caution, intended for developers only. | */
15:    public static final int MAV_MODE_STABILIZE_ARMED = 208; /* System is allowe
d to be active, under assisted RC control. | */
16:    public static final int MAV_MODE_GUIDED_ARMED = 216; /* System is allowed
to be active, under autonomous control, manual setpoint | */
17:    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 an
d not pre-programmed by MISSIONs) | */
18:    public static final int MAV_MODE_ENUM_END = 221; /* | */
19: }

```



```

1: /** These flags encode the MAV mode.
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_MODE_FLAG {
6:     public static final int MAV_MODE_FLAG_CUSTOM_MODE_ENABLED = 1; /* 0b000000
01 Reserved for future use. | */
7:     public static final int MAV_MODE_FLAG_TEST_ENABLED = 2; /* 0b00000010 syst
em has a test mode enabled. This flag is intended for temporary system tests and should n
ot be used for stable implementations. | */
8:     public static final int MAV_MODE_FLAG_AUTO_ENABLED = 4; /* 0b00000100 auto
nomous mode enabled, system finds its own goal positions. Guided flag can be set or not,
depends on the actual implementation. | */
9:     public static final int MAV_MODE_FLAG_GUIDED_ENABLED = 8; /* 0b00001000 gu
ided mode enabled, system flies MISSIONs / mission items. | */
10:    public static final int MAV_MODE_FLAG_STABILIZE_ENABLED = 16; /* 0b0001000
0 system stabilizes electronically its attitude (and optionally position). It needs howev
er further control inputs to move around. | */
11:    public static final int MAV_MODE_FLAG_HIL_ENABLED = 32; /* 0b00100000 hard
ware in the loop simulation. All motors / actuators are blocked, but internal software is
full operational. | */
12:    public static final int MAV_MODE_FLAG_MANUAL_INPUT_ENABLED = 64; /* 0b0100
0000 remote control input is enabled. | */
13:    public static final int MAV_MODE_FLAG_SAFETY_ARMED = 128; /* 0b10000000 MA
V safety set to armed. Motors are enabled / running / can start. Ready to fly. | */
14:    public static final int MAV_MODE_FLAG_ENUM_END = 129; /* | */
15: }

```



```

1: /** 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.
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_MODE_FLAG_DECODE_POSITION {
6:     public static final int MAV_MODE_FLAG_DECODE_POSITION_CUSTOM_MODE = 1; /*
Eighth bit: 00000001 | */
7:     public static final int MAV_MODE_FLAG_DECODE_POSITION_TEST = 2; /* Seventh
bit: 00000010 | */
8:     public static final int MAV_MODE_FLAG_DECODE_POSITION_AUTO = 4; /* Sixth
bit: 00000100 | */
9:     public static final int MAV_MODE_FLAG_DECODE_POSITION_GUIDED = 8; /* Fifth
bit: 00001000 | */
10:    public static final int MAV_MODE_FLAG_DECODE_POSITION_STABILIZE = 16; /* Fourth
bit: 00010000 | */
11:    public static final int MAV_MODE_FLAG_DECODE_POSITION_HIL = 32; /* Third
bit: 00100000 | */
12:    public static final int MAV_MODE_FLAG_DECODE_POSITION_MANUAL = 64; /* Second
bit: 01000000 | */
13:    public static final int MAV_MODE_FLAG_DECODE_POSITION_SAFETY = 128; /* First
bit: 10000000 | */
14:    public static final int MAV_MODE_FLAG_DECODE_POSITION_ENUM_END = 129; /*
| */
15: }

```



```

1: /** Enumeration of possible mount operation modes
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_MOUNT_MODE {
6:     public static final int MAV_MOUNT_MODE_RETRACT = 0; /* Load and keep safe
position (Roll,Pitch,Yaw) from EEPROM and stop stabilization | */
7:     public static final int MAV_MOUNT_MODE_NEUTRAL = 1; /* Load and keep neutr
al position (Roll,Pitch,Yaw) from EEPROM. | */
8:     public static final int MAV_MOUNT_MODE_MAVLINK_TARGETING = 2; /* Load neut
ral position and start MAVLink Roll,Pitch,Yaw control with stabilization | */
9:     public static final int MAV_MOUNT_MODE_RC_TARGETING = 3; /* Load neutral p
osition and start RC Roll,Pitch,Yaw control with stabilization | */
10:    public static final int MAV_MOUNT_MODE_GPS_POINT = 4; /* Load neutral posi
tion and start to point to Lat,Lon,Alt | */
11:    public static final int MAV_MOUNT_MODE_ENUM_END = 5; /* | */
12: }

```



```
1: /** Specifies the datatype of a MAVLink parameter.
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_PARAM_TYPE {
6:     public static final int MAV_PARAM_TYPE_UINT8 = 1; /* 8-bit unsigned integer
r | */
7:     public static final int MAV_PARAM_TYPE_INT8 = 2; /* 8-bit signed integer |
*/
8:     public static final int MAV_PARAM_TYPE_UINT16 = 3; /* 16-bit unsigned integer
ger | */
9:     public static final int MAV_PARAM_TYPE_INT16 = 4; /* 16-bit signed integer
| */
10:    public static final int MAV_PARAM_TYPE_UINT32 = 5; /* 32-bit unsigned integer
ger | */
11:    public static final int MAV_PARAM_TYPE_INT32 = 6; /* 32-bit signed integer
| */
12:    public static final int MAV_PARAM_TYPE_UINT64 = 7; /* 64-bit unsigned integer
ger | */
13:    public static final int MAV_PARAM_TYPE_INT64 = 8; /* 64-bit signed integer
| */
14:    public static final int MAV_PARAM_TYPE_REAL32 = 9; /* 32-bit floating-point
t | */
15:    public static final int MAV_PARAM_TYPE_REAL64 = 10; /* 64-bit floating-point
nt | */
16:    public static final int MAV_PARAM_TYPE_ENUM_END = 11; /* | */
17: }
```



```

1: /** result from a mavlink command
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_RESULT {
6:     public static final int MAV_RESULT_ACCEPTED = 0; /* Command ACCEPTED and E
EXECUTED | */
7:     public static final int MAV_RESULT_TEMPORARILY_REJECTED = 1; /* Command TE
MPORARY REJECTED/DENIED | */
8:     public static final int MAV_RESULT_DENIED = 2; /* Command PERMANENTLY DENI
ED | */
9:     public static final int MAV_RESULT_UNSUPPORTED = 3; /* Command UNKNOWN/UNS
UPPORTED | */
10:    public static final int MAV_RESULT_FAILED = 4; /* Command executed, but fa
iled | */
11:    public static final int MAV_RESULT_ENUM_END = 5; /* | */
12: }

```



```

1: /** The ROI (region of interest) for the vehicle. This can be
2:      be used by the vehicle for camera/vehicle attitude alignment (see
3:      MAV_CMD_NAV_ROI).
4: */
5: package com.MAVLink.Messages.enums;
6:
7: public class MAV_ROI {
8:     public static final int MAV_ROI_NONE = 0; /* No region of interest. | */
9:     public static final int MAV_ROI_WPNEXT = 1; /* Point toward next MISSION.
| */
10:    public static final int MAV_ROI_WPINDEX = 2; /* Point toward given MISSION
. | */
11:    public static final int MAV_ROI_LOCATION = 3; /* Point toward fixed locati
on. | */
12:    public static final int MAV_ROI_TARGET = 4; /* Point toward of given id. |
*/
13:    public static final int MAV_ROI_ENUM_END = 5; /* | */
14: }

```



```

1: /** Indicates the severity level, generally used for status messages to indicate t
heir relative urgency. Based on RFC-5424 using expanded definitions at: http://www.kiwisyslog.com/kb/info:-syslog-message-levels/.
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_SEVERITY {
6:     public static final int MAV_SEVERITY_EMERGENCY = 0; /* System is unusable.
This is a "panic" condition. | */
7:     public static final int MAV_SEVERITY_ALERT = 1; /* Action should be taken
immediately. Indicates error in non-critical systems. | */
8:     public static final int MAV_SEVERITY_CRITICAL = 2; /* Action must be taken
immediately. Indicates failure in a primary system. | */
9:     public static final int MAV_SEVERITY_ERROR = 3; /* Indicates an error in s
econdary/redundant systems. | */
10:    public static final int MAV_SEVERITY_WARNING = 4; /* Indicates about a pos
sible future error if this is not resolved within a given timeframe. Example would be a l
ow battery warning. | */
11:    public static final int MAV_SEVERITY_NOTICE = 5; /* An unusual event has o
ccured, though not an error condition. This should be investigated for the root cause. |
*/
12:    public static final int MAV_SEVERITY_INFO = 6; /* Normal operational messa
ges. Useful for logging. No action is required for these messages. | */
13:    public static final int MAV_SEVERITY_DEBUG = 7; /* Useful non-operational
messages that can assist in debugging. These should not occur during normal operation. |
*/
14:    public static final int MAV_SEVERITY_ENUM_END = 8; /* | */
15: }

```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_STATE {
6:     public static final int MAV_STATE_UNINIT = 0; /* Uninitialized system, sta
te is unknown. | */
7:     public static final int MAV_STATE_BOOT = 1; /* System is booting up. | */
8:     public static final int MAV_STATE_CALIBRATING = 2; /* System is calibratin
g and not flight-ready. | */
9:     public static final int MAV_STATE_STANDBY = 3; /* System is grounded and o
n standby. It can be launched any time. | */
10:    public static final int MAV_STATE_ACTIVE = 4; /* System is active and migh
t be already airborne. Motors are engaged. | */
11:    public static final int MAV_STATE_CRITICAL = 5; /* System is in a non-norm
al flight mode. It can however still navigate. | */
12:    public static final int MAV_STATE_EMERGENCY = 6; /* System is in a non-nor
mal flight mode. It lost control over parts or over the whole airframe. It is in mayday a
nd going down. | */
13:    public static final int MAV_STATE_POWEROFF = 7; /* System just initialized
its power-down sequence, will shut down now. | */
14:    public static final int MAV_STATE_ENUM_END = 8; /* | */
15: }
```



```
1: /**
2: */
3: package com.MAVLink.Messages.enums;
4:
5: public class MAV_TYPE {
6:     public static final int MAV_TYPE_GENERIC = 0; /* Generic micro air vehicle
. | */
7:     public static final int MAV_TYPE_FIXED_WING = 1; /* Fixed wing aircraft. |
*/
8:     public static final int MAV_TYPE_QUADROTOR = 2; /* Quadrotor | */
9:     public static final int MAV_TYPE_COAXIAL = 3; /* Coaxial helicopter | */
10:    public static final int MAV_TYPE_HELICOPTER = 4; /* Normal helicopter with
tail rotor. | */
11:    public static final int MAV_TYPE_ANTENNA_TRACKER = 5; /* Ground installati
on | */
12:    public static final int MAV_TYPE_GCS = 6; /* Operator control unit / groun
d control station | */
13:    public static final int MAV_TYPE_AIRSHIP = 7; /* Airship, controlled | */
14:    public static final int MAV_TYPE_FREE_BALLOON = 8; /* Free balloon, uncont
rolled | */
15:    public static final int MAV_TYPE_ROCKET = 9; /* Rocket | */
16:    public static final int MAV_TYPE_GROUND_ROVER = 10; /* Ground rover | */
17:    public static final int MAV_TYPE_SURFACE_BOAT = 11; /* Surface vessel, boa
t, ship | */
18:    public static final int MAV_TYPE_SUBMARINE = 12; /* Submarine | */
19:    public static final int MAV_TYPE_HEXAROTOR = 13; /* Hexarotor | */
20:    public static final int MAV_TYPE_OCTOROTOR = 14; /* Octorotor | */
21:    public static final int MAV_TYPE_TRICOPTER = 15; /* Octorotor | */
22:    public static final int MAV_TYPE_FLAPPING_WING = 16; /* Flapping wing | */
23:    public static final int MAV_TYPE_KITE = 17; /* Flapping wing | */
24:    public static final int MAV_TYPE_ENUM_END = 18; /* | */
25: }
```



```

1:
2: package com.MAVLink.Messages;
3:
4: import java.io.Serializable;
5:
6: public class MAVLinkMessage implements Serializable {
7:     private static final long serialVersionUID = -7754622750478538539L;
8:     // The MAVLink message classes have been changed to implement Serializable
9:
10: // this way is possible to pass a mavlink message through the Service-Activity interface
11: /**
12:  * Simply a common interface for all MAVLink Messages
13:  */
14:
15: public int sysid;
16: public int compid;
17: public int msgid;
18:
19: }
20:

```



```

1: package com.MAVLink.Messages;
2:
3: import android.util.Log;
4: import java.io.Serializable;
5: import com.MAVLink.Messages.ardupilotmega.*;
6:
7: /**
8:  * Common interface for all MAVLink Messages
9:  * Packet Anatomy
10:  * This is the anatomy of one packet. It is inspired by the CAN and SAE AS-4 stand
ards.
11:
12:  * Byte Index  Content          Value      Explanation
13:  * 0           Packet start sign v1.0: 0xFE Indicates the start of a new packe m
t. (v0.9: 0x55)
14:  * 1           Payload length    0 - 255    Indicates length of the following f
payload.
15:  * 2           Packet sequence   0 - 255    Each component counts up his send
sequence. Allows to detect packet loss
16:  * 3           System ID        1 - 255    ID of the SENDING system. Allows t
o differentiate different MAVs on the same network.
17:  * 4           Component ID      0 - 255    ID of the SENDING component. Allow
s to differentiate different components of the same system, e.g. the IMU and the autopilo
t.
18:  * 5           Message ID       0 - 255    ID of the message - the id defines
what the payload means and how it should be correctly decoded.
19:  * 6 to (n+6) Payload          0 - 255    Data of the message, depends on th
e message id.
20:  * (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 versio
n of the same packet but with different variables).
21:
22:  * The checksum is the same as used in ITU X.25 and SAE AS-4 standards (CRC-16-CCI
TT), documented in SAE AS5669A. Please see the MAVLink source code for a documented C-imp
lementation of it. LINK TO CHECKSUM
23:  * The minimum packet length is 8 bytes for acknowledgement packets without payloa
d
24:  * The maximum packet length is 263 bytes for full payload
25:  *
26:  * @author ghelle
27:  *
28:  */
29: public class MAVLinkPacket implements Serializable {
30:     private static final long serialVersionUID = 2095947771227815314L;
31:
32:     public static final int MAVLINK_STX = 254;
33:
34:     /**
35:      * Message length. NOT counting STX, LENGTH, SEQ, SYSID, COMPID, MSGID, CR
C1 and CRC2
36:      */
37:     public int len;
38:     /**
39:      * Message sequence
40:      */
41:     public int seq;
42:     /**
43:      * ID of the SENDING system. Allows to differentiate different MAVs on the
44:      * same network.
45:      */
46:     public int sysid;
47:     /**
48:      * ID of the SENDING component. Allows to differentiate different componen
ts
49:      * of the same system, e.g. the IMU and the autopilot.
50:      */
51:
52:     public int compid;
53:     /**
54:      * ID of the message - the id defines what the payload means and how it
55:      * should be correctly decoded.
56:      */
57:     public int msgid;
58:     /**
59:      * Data of the message, depends on the message id.
60:      */
61:     public MAVLinkPayload payload;
62:     /**
63:      * ITU X.25/SAE AS-4 hash, excluding packet start sign, so bytes 1..(n+6)
64:      * Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed fro
65:      * message fields. Protects the packet from decoding a different version o
66:      * the same packet but with different variables).
67:      */
68:     public CRC crc;
69:
70:     public MAVLinkPacket(){
71:         payload = new MAVLinkPayload();
72:     }
73:
74:     /**
75:      * Check if the size of the Payload is equal to the "len" byte
76:      */
77:     public boolean payloadIsFilled() {
78:         if (payload.size() >= MAVLinkPayload.MAX_PAYLOAD_SIZE-1) {
79:             Log.d("MAV", "Buffer overflow");
80:             return true;
81:         }
82:         return (payload.size() == len);
83:     }
84:
85:     /**
86:      * Update CRC for this packet.
87:      */
88:     public void generateCRC(){
89:         crc = new CRC();
90:         crc.update_checksum(len);
91:         crc.update_checksum(seq);
92:         crc.update_checksum(sysid);
93:         crc.update_checksum(compid);
94:         crc.update_checksum(msgid);
95:         payload.resetIndex();
96:         for (int i = 0; i < payload.size(); i++) {
97:             crc.update_checksum(payload.getBytes(i));
98:         }
99:         crc.finish_checksum(msgid);
100:     }
101:
102:     /**
103:      * Encode this packet for transmission.
104:      *
105:      * @return Array with bytes to be transmitted
106:      */
107:     public byte[] encodePacket() {
108:         byte[] buffer = new byte[6 + len + 2];
109:         int i = 0;
110:         buffer[i++] = (byte) MAVLINK_STX;
111:         buffer[i++] = (byte) len;
112:         buffer[i++] = (byte) seq;
113:         buffer[i++] = (byte) sysid;
114:         buffer[i++] = (byte) compid;
115:         buffer[i++] = (byte) msgid;
116:         for (int j = 0; j < payload.size(); j++) {

```

```

116:         buffer[i++] = payload.payload.get(j);
117:     }
118:     generateCRC();
119:     buffer[i++] = (byte) (crc.getLSB());
120:     buffer[i++] = (byte) (crc.getMSB());
121:     return buffer;
122: }
123:
124: /**
125:  * Unpack the data in this packet and return a MAVLink message
126:  *
127:  * @return MAVLink message decoded from this packet
128:  */
129: public MAVLinkMessage unpack() {
130:     switch (msgid) {
131:         case msg_sensor_offsets.MAVLINK_MSG_ID_SENSOR_OFFSETS:
132:             return new msg_sensor_offsets(this);
133:         case msg_set_mag_offsets.MAVLINK_MSG_ID_SET_MAG_OFFSETS:
134:             return new msg_set_mag_offsets(this);
135:         case msg_meminfo.MAVLINK_MSG_ID_MEMINFO:
136:             return new msg_meminfo(this);
137:         case msg_ap_adc.MAVLINK_MSG_ID_AP_ADC:
138:             return new msg_ap_adc(this);
139:         case msg_digicam_configure.MAVLINK_MSG_ID_DIGICAM_CONFIGURE:
140:             return new msg_digicam_configure(this);
141:         case msg_digicam_control.MAVLINK_MSG_ID_DIGICAM_CONTROL:
142:             return new msg_digicam_control(this);
143:         case msg_mount_configure.MAVLINK_MSG_ID_MOUNT_CONFIGURE:
144:             return new msg_mount_configure(this);
145:         case msg_mount_control.MAVLINK_MSG_ID_MOUNT_CONTROL:
146:             return new msg_mount_control(this);
147:         case msg_mount_status.MAVLINK_MSG_ID_MOUNT_STATUS:
148:             return new msg_mount_status(this);
149:         case msg_fence_point.MAVLINK_MSG_ID_FENCE_POINT:
150:             return new msg_fence_point(this);
151:         case msg_fence_fetch_point.MAVLINK_MSG_ID_FENCE_FETCH_POINT:
152:             return new msg_fence_fetch_point(this);
153:         case msg_fence_status.MAVLINK_MSG_ID_FENCE_STATUS:
154:             return new msg_fence_status(this);
155:         case msg_ahrs.MAVLINK_MSG_ID_AHRS:
156:             return new msg_ahrs(this);
157:         case msg_simstate.MAVLINK_MSG_ID_SIMSTATE:
158:             return new msg_simstate(this);
159:         case msg_hwstatus.MAVLINK_MSG_ID_HWSTATUS:
160:             return new msg_hwstatus(this);
161:         case msg_radio.MAVLINK_MSG_ID_RADIO:
162:             return new msg_radio(this);
163:         case msg_limits_status.MAVLINK_MSG_ID_LIMITS_STATUS:
164:             return new msg_limits_status(this);
165:         case msg_wind.MAVLINK_MSG_ID_WIND:
166:             return new msg_wind(this);
167:         case msg_data16.MAVLINK_MSG_ID_DATA16:
168:             return new msg_data16(this);
169:         case msg_data32.MAVLINK_MSG_ID_DATA32:
170:             return new msg_data32(this);
171:         case msg_data64.MAVLINK_MSG_ID_DATA64:
172:             return new msg_data64(this);
173:         case msg_data96.MAVLINK_MSG_ID_DATA96:
174:             return new msg_data96(this);
175:         case msg_heartbeat.MAVLINK_MSG_ID_HEARTBEAT:
176:             return new msg_heartbeat(this);
177:         case msg_sys_status.MAVLINK_MSG_ID_SYS_STATUS:
178:             return new msg_sys_status(this);
179:         case msg_system_time.MAVLINK_MSG_ID_SYSTEM_TIME:
180:             return new msg_system_time(this);
181:         case msg_ping.MAVLINK_MSG_ID_PING:
182:             return new msg_ping(this);

```

```

183:     NTROL:
184:     184:
185:     185:
186:     R_CONTROL_ACK:
187:     186:
188:     187:
189:     188:
190:     189:
191:     190:
192:     191:
193:     192:
194:     193:
195:     194:
196:     195:
197:     196:
198:     197:
199:     198:
200:     199:
201:     200:
202:     201:
203:     202:
204:     203:
205:     204:
206:     205:
207:     206:
208:     207:
209:     208:
210:     209:
211:     210:
212:     211:
213:     212:
214:     213:
215:     214:
216:     215:
217:     216:
218:     217:
219:     218:
220:     219:
221:     220:
222:     221:
223:     222:
224:     223:
225:     224:
226:     ST_PARTIAL_LIST:
227:     225:
228:     ARTIAL_LIST:
229:     226:
230:     227:
231:     228:
232:     229:
233:     230:
234:     231:
235:     232:
236:     233:
237:     234:
238:     235:
239:     236:
240:     237:
241:     238:
242:     239:
243:     240:
244:     241:
245:     242:

case msg_change_operator_control.MAVLINK_MSG_ID_CHANGE_OPERATOR_CO

    return new msg_change_operator_control(this);
case msg_change_operator_control_ack.MAVLINK_MSG_ID_CHANGE_OPERATO

    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_REQUE

    return new msg_mission_request_partial_list(this);
case msg_mission_write_partial_list.MAVLINK_MSG_ID_MISSION_WRITE_P

    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:

```

```

246:         return new msg_mission_ack(this);
247:     case msg_set_gps_global_origin.MAVLINK_MSG_ID_SET_GPS_GLOBAL_ORIGI
N:
248:         return new msg_set_gps_global_origin(this);
249:     case msg_gps_global_origin.MAVLINK_MSG_ID_GPS_GLOBAL_ORIGIN:
250:         return new msg_gps_global_origin(this);
251:     case msg_set_local_position_setpoint.MAVLINK_MSG_ID_SET_LOCAL_POSI
TION_SETPOINT:
252:         return new msg_set_local_position_setpoint(this);
253:     case msg_local_position_setpoint.MAVLINK_MSG_ID_LOCAL_POSITION_SET
POINT:
254:         return new msg_local_position_setpoint(this);
255:     case msg_global_position_setpoint_int.MAVLINK_MSG_ID_GLOBAL_POSITI
ON_SETPOINT_INT:
256:         return new msg_global_position_setpoint_int(this);
257:     case msg_set_global_position_setpoint_int.MAVLINK_MSG_ID_SET_GLOBA
L_POSITION_SETPOINT_INT:
258:         return new msg_set_global_position_setpoint_int(this);
259:     case msg_safety_set_allowed_area.MAVLINK_MSG_ID_SAFETY_SET_ALLOWED
_AREA:
260:         return new msg_safety_set_allowed_area(this);
261:     case msg_safety_allowed_area.MAVLINK_MSG_ID_SAFETY_ALLOWED_AREA:
262:         return new msg_safety_allowed_area(this);
263:     case msg_set_roll_pitch_yaw_thrust.MAVLINK_MSG_ID_SET_ROLL_PITCH_Y
AW_THRUST:
264:         return new msg_set_roll_pitch_yaw_thrust(this);
265:     case msg_set_roll_pitch_yaw_speed_thrust.MAVLINK_MSG_ID_SET_ROLL_P
ITCH_YAW_SPEED_THRUST:
266:         return new msg_set_roll_pitch_yaw_speed_thrust(this);
267:     case msg_roll_pitch_yaw_thrust_setpoint.MAVLINK_MSG_ID_ROLL_PITCH_
YAW_THRUST_SETPOINT:
268:         return new msg_roll_pitch_yaw_thrust_setpoint(this);
269:     case msg_roll_pitch_yaw_speed_thrust_setpoint.MAVLINK_MSG_ID_ROLL_
PITCH_YAW_SPEED_THRUST_SETPOINT:
270:         return new msg_roll_pitch_yaw_speed_thrust_setpoint(this);
;
271:     case msg_set_quad_motors_setpoint.MAVLINK_MSG_ID_SET_QUAD_MOTORS_S
ETPOINT:
272:         return new msg_set_quad_motors_setpoint(this);
273:     case msg_set_quad_swarm_roll_pitch_yaw_thrust.MAVLINK_MSG_ID_SET_Q
UAD_SWARM_ROLL_PITCH_YAW_THRUST:
274:         return new msg_set_quad_swarm_roll_pitch_yaw_thrust(this);
;
275:     case msg_nav_controller_output.MAVLINK_MSG_ID_NAV_CONTROLLER_OUTPU
T:
276:         return new msg_nav_controller_output(this);
277:     case msg_set_quad_swarm_led_roll_pitch_yaw_thrust.MAVLINK_MSG_ID_S
ET_QUAD_SWARM_LED_ROLL_PITCH_YAW_THRUST:
278:         return new msg_set_quad_swarm_led_roll_pitch_yaw_thrust(t
his);
279:     case msg_state_correction.MAVLINK_MSG_ID_STATE_CORRECTION:
280:         return new msg_state_correction(this);
281:     case msg_request_data_stream.MAVLINK_MSG_ID_REQUEST_DATA_STREAM:
282:         return new msg_request_data_stream(this);
283:     case msg_data_stream.MAVLINK_MSG_ID_DATA_STREAM:
284:         return new msg_data_stream(this);
285:     case msg_manual_control.MAVLINK_MSG_ID_MANUAL_CONTROL:
286:         return new msg_manual_control(this);
287:     case msg_rc_channels_override.MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE:
288:         return new msg_rc_channels_override(this);
289:     case msg_vfr_hud.MAVLINK_MSG_ID_VFR_HUD:
290:         return new msg_vfr_hud(this);
291:     case msg_command_long.MAVLINK_MSG_ID_COMMAND_LONG:
292:         return new msg_command_long(this);
293:     case msg_command_ack.MAVLINK_MSG_ID_COMMAND_ACK:
294:         return new msg_command_ack(this);
295:     case msg_roll_pitch_yaw_rates_thrust_setpoint.MAVLINK_MSG_ID_ROLL_
PITCH_YAW_RATES_THRUST_SETPOINT:
296:         return new msg_roll_pitch_yaw_rates_thrust_setpoint(this);
;
297:     case msg_manual_setpoint.MAVLINK_MSG_ID_MANUAL_SETPOINT:
298:         return new msg_manual_setpoint(this);
299:     case msg_local_position_ned_system_global_offset.MAVLINK_MSG_ID_LO
CAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET:
300:         return new msg_local_position_ned_system_global_offset(th
is);
301:     case msg_hil_state.MAVLINK_MSG_ID_HIL_STATE:
302:         return new msg_hil_state(this);
303:     case msg_hil_controls.MAVLINK_MSG_ID_HIL_CONTROLS:
304:         return new msg_hil_controls(this);
305:     case msg_hil_rc_inputs_raw.MAVLINK_MSG_ID_HIL_RC_INPUTS_RAW:
306:         return new msg_hil_rc_inputs_raw(this);
307:     case msg_optical_flow.MAVLINK_MSG_ID_OPTICAL_FLOW:
308:         return new msg_optical_flow(this);
309:     case msg_global_vision_position_estimate.MAVLINK_MSG_ID_GLOBAL_VIS
ION_POSITION_ESTIMATE:
310:         return new msg_global_vision_position_estimate(this);
311:     case msg_vision_position_estimate.MAVLINK_MSG_ID_VISION_POSITION_E
STIMATE:
312:         return new msg_vision_position_estimate(this);
313:     case msg_vision_speed_estimate.MAVLINK_MSG_ID_VISION_SPEED_ESTIMAT
E:
314:         return new msg_vision_speed_estimate(this);
315:     case msg_vicon_position_estimate.MAVLINK_MSG_ID_VICON_POSITION_EST
IMATE:
316:         return new msg_vicon_position_estimate(this);
317:     case msg_highres_imu.MAVLINK_MSG_ID_HIGHRES_IMU:
318:         return new msg_highres_imu(this);
319:     case msg_file_transfer_start.MAVLINK_MSG_ID_FILE_TRANSFER_START:
320:         return new msg_file_transfer_start(this);
321:     case msg_file_transfer_dir_list.MAVLINK_MSG_ID_FILE_TRANSFER_DIR_L
IST:
322:         return new msg_file_transfer_dir_list(this);
323:     case msg_file_transfer_res.MAVLINK_MSG_ID_FILE_TRANSFER_RES:
324:         return new msg_file_transfer_res(this);
325:     case msg_battery_status.MAVLINK_MSG_ID_BATTERY_STATUS:
326:         return new msg_battery_status(this);
327:     case msg_setpoint_8dof.MAVLINK_MSG_ID_SETPOINT_8DOF:
328:         return new msg_setpoint_8dof(this);
329:     case msg_setpoint_6dof.MAVLINK_MSG_ID_SETPOINT_6DOF:
330:         return new msg_setpoint_6dof(this);
331:     case msg_memory_vect.MAVLINK_MSG_ID_MEMORY_VECT:
332:         return new msg_memory_vect(this);
333:     case msg_debug_vect.MAVLINK_MSG_ID_DEBUG_VECT:
334:         return new msg_debug_vect(this);
335:     case msg_named_value_float.MAVLINK_MSG_ID_NAMED_VALUE_FLOAT:
336:         return new msg_named_value_float(this);
337:     case msg_named_value_int.MAVLINK_MSG_ID_NAMED_VALUE_INT:
338:         return new msg_named_value_int(this);
339:     case msg_statustext.MAVLINK_MSG_ID_STATUSTEXT:
340:         return new msg_statustext(this);
341:     case msg_debug.MAVLINK_MSG_ID_DEBUG:
342:         return new msg_debug(this);
default:
343:     Log.d("MAVLink", "UNKNOWN MESSAGE - " + msgid);
344:     return null;
}
}
345:
346:
347:
348:
349:
350:

```



```

1: package com.MAVLink.Messages;
2:
3: import java.nio.ByteBuffer;
4:
5: public class MAVLinkPayload {
6:
7:     public static final int MAX_PAYLOAD_SIZE = 512;
8:
9:     public ByteBuffer payload;
10:    public int index;
11:
12:    public MAVLinkPayload() {
13:        payload = ByteBuffer.allocate(MAX_PAYLOAD_SIZE);
14:    }
15:
16:    public ByteBuffer getData() {
17:        return payload;
18:    }
19:
20:    public int size() {
21:        return payload.position();
22:    }
23:
24:    public void add(byte c) {
25:        payload.put(c);
26:    }
27:
28:    public void resetIndex() {
29:        index = 0;
30:    }
31:
32:    public byte getByte() {
33:        byte result = 0;
34:        result |= (payload.get(index + 0) & 0xFF);
35:        index += 1;
36:        return (byte) result;
37:    }
38:
39:    public short getShort() {
40:        short result = 0;
41:        result |= (payload.get(index + 1) & 0xFF) << 8;
42:        result |= (payload.get(index + 0) & 0xFF);
43:        index += 2;
44:        return (short) result;
45:    }
46:
47:    public int getInt() {
48:        int result = 0;
49:        result |= (payload.get(index + 3) & 0xFF) << 24;
50:        result |= (payload.get(index + 2) & 0xFF) << 16;
51:        result |= (payload.get(index + 1) & 0xFF) << 8;
52:        result |= (payload.get(index + 0) & 0xFF);
53:        index += 4;
54:        return (int) result;
55:    }
56:
57:    public long getLong() {
58:        long result = 0;
59:        result |= (payload.get(index + 7) & 0xFF) << 56;
60:        result |= (payload.get(index + 6) & 0xFF) << 48;
61:        result |= (payload.get(index + 5) & 0xFF) << 40;
62:        result |= (payload.get(index + 4) & 0xFF) << 32;
63:        result |= (payload.get(index + 3) & 0xFF) << 24;
64:        result |= (payload.get(index + 2) & 0xFF) << 16;
65:        result |= (payload.get(index + 1) & 0xFF) << 8;
66:        result |= (payload.get(index + 0) & 0xFF);
67:        index += 8;

```

```

68:        return (long) result;
69:    }
70:
71:    public float getFloat() {
72:        return Float.intBitsToFloat(getInt());
73:    }
74:
75:    public void putByte(byte data) {
76:        add(data);
77:    }
78:
79:    public void putShort(short data) {
80:        add((byte) (data >> 0));
81:        add((byte) (data >> 8));
82:    }
83:
84:    public void putInt(int data) {
85:        add((byte) (data >> 0));
86:        add((byte) (data >> 8));
87:        add((byte) (data >> 16));
88:        add((byte) (data >> 24));
89:    }
90:
91:    public void putLong(long data) {
92:        add((byte) (data >> 0));
93:        add((byte) (data >> 8));
94:        add((byte) (data >> 16));
95:        add((byte) (data >> 24));
96:        add((byte) (data >> 32));
97:        add((byte) (data >> 40));
98:        add((byte) (data >> 48));
99:        add((byte) (data >> 56));
100:    }
101:
102:    public void putFloat(float data) {
103:        putInt(Float.floatToIntBits(data));
104:    }
105:
106: }

```



```

1: package com.MAVLink.Messages;
2:
3:
4: /**
5:  * Storage for MAVLink Packet and Error statistics
6:  *
7:  * @author Helibot
8:  *
9:  */
10: public class MAVLinkStats /* implements Serializable */{
11:
12:     public int receivedPacketCount;
13:
14:     public int crcErrorCount;
15:
16:     public int lostPacketCount;
17:
18:     private int lastPacketSeq;
19:
20:     /**
21:      * Check the new received packet to see if has lost someone between this a
22:      * the last packet
23:      *
24:      * @param packet
25:      *     Packet that should be checked
26:      */
27:     public void newPacket(MAVLinkPacket packet) {
28:         advanceLastPacketSequence();
29:         if (hasLostPackets(packet)) {
30:             updateLostPacketCount(packet);
31:         }
32:         lastPacketSeq = packet.seq;
33:         receivedPacketCount++;
34:     }
35:
36:     private void updateLostPacketCount(MAVLinkPacket packet) {
37:         int lostPackets;
38:         if (packet.seq - lastPacketSeq < 0) {
39:             lostPackets = (packet.seq - lastPacketSeq) + 255;
40:         } else {
41:             lostPackets = (packet.seq - lastPacketSeq);
42:         }
43:         lostPacketCount += lostPackets;
44:     }
45:
46:     private boolean hasLostPackets(MAVLinkPacket packet) {
47:         return lastPacketSeq > 0 && packet.seq != lastPacketSeq;
48:     }
49:
50:     private void advanceLastPacketSequence() {
51:         // wrap from 255 to 0 if necessary
52:         lastPacketSeq = (lastPacketSeq + 1) & 0xFF;
53:     }
54:
55:     /**
56:      * Called when a CRC error happens on the parser
57:      */
58:     public void crcError() {
59:         crcErrorCount++;
60:     }
61:
62:     /**
63:      * Resets statistics for this MAVLink.
64:      */
65:     public void mavlinkResetStats() {
66:         lastPacketSeq = -1;
67:         lostPacketCount = 0;
68:         crcErrorCount = 0;
69:         receivedPacketCount = 0;
70:     }
71:
72: }

```



```

1: package com.MAVLink;
2:
3: import com.MAVLink.Messages.MAVLinkPacket;
4: import com.MAVLink.Messages.MAVLinkStats;
5:
6: public class Parser {
7:
8:     /**
9:      * States from the parsing state machine
10:    */
11:    enum MAV_states {
12:        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
13:    }
14:
15:    MAV_states state = MAV_states.MAVLINK_PARSE_STATE_UNINIT;
16:
17:    static boolean msg_received;
18:
19:    public MAVLinkStats stats = new MAVLinkStats();
20:    private MAVLinkPacket m;
21:
22:    /**
23:     * This is a convenience function which handles the complete MAVLink
24:     * parsing. the function will parse one byte at a time and return the
25:     * complete packet once it could be successfully decoded. Checksum and other
26:     * failures will be silently ignored.
27:     *
28:     * @param c
29:     *      The char to parse
30:     */
31:    public MAVLinkPacket mavlink_parse_char(int c) {
32:        msg_received = false;
33:
34:        switch (state) {
35:            case MAVLINK_PARSE_STATE_UNINIT:
36:            case MAVLINK_PARSE_STATE_IDLE:
37:
38:                if (c == MAVLinkPacket.MAVLINK_STX) {
39:                    state = MAV_states.MAVLINK_PARSE_STATE_GOT_STX;
40:                    m = new MAVLinkPacket();
41:                }
42:                break;
43:
44:            case MAVLINK_PARSE_STATE_GOT_STX:
45:                if (msg_received) {
46:                    msg_received = false;
47:                    state = MAV_states.MAVLINK_PARSE_STATE_IDLE;
48:                } else {
49:                    m.len = c;
50:                    state = MAV_states.MAVLINK_PARSE_STATE_GOT_LENGTH;
51:                }
52:                break;
53:
54:            case MAVLINK_PARSE_STATE_GOT_LENGTH:
55:                m.seq = c;
56:                state = MAV_states.MAVLINK_PARSE_STATE_GOT_SEQ;
57:                break;
58:
59:            case MAVLINK_PARSE_STATE_GOT_SEQ:
60:                m.sysid = c;
61:                state = MAV_states.MAVLINK_PARSE_STATE_GOT_SYSID;
62:                break;
63:

```

```

64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:

```

```

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;
    }
}

if (msg_received) {
    return m;
} else {
    return null;
}

```

./com/MAVLink/Parser.java

Fri Oct 25 14:10:51 2013

2

```
127:
128:
129: }
```