The following is the description of skysense protocol between single-board computer (API server) and any GUI or service (API client) which wants to control charging station and wants to receive charging state as events.

Skysense protocol is based on ZeroMQ distributed messaging http://zeromq.org/ library, which works on top of the TCP/IP stack and provides required communication patterns as REQUEST-RESPONSE, SUBSCRIBE-PUBLISH. ZeroMQ protocol is implemented on variety of languages (C, C++, Java, Python, etc) and thus gives a lot of freedom to change implementation without changing the protocol.

Skysense protocol consists of two parts: charging control commands, which follow REQUEST-RESPONSE pattern and charging events, which follow SUBSCRIBE-PUBLISH pattern.

NOTE: all integers are presented in little-endian format.

Protocol enumerators

```
enum sky_dev_type {
    SKY_INDOOR  = 0,
    SKY_OUTDOOR = 1,
};

enum sky_dev_param {
    SKY_EEPROM_INITED            = 0,
    SKY_SCANNING_INTERVAL        = 1,
    SKY_PRECHARGING_INTERVAL     = 2,
    SKY_PRECHARGING_COUNTER      = 3,
    SKY_POSTCHARGING_INTERVAL    = 4,
    SKY_POSTCHARGING_DELAY       = 5,
    SKY_WET_DELAY                = 6,
    SKY_SHORTCIRC_DELAY          = 7,
    SKY_THRESH_FINISH_CHARGING   = 8,
    SKY_THRESH_NOCHARGER_PRESENT = 9,
    SKY_THRESH_SHORTCIRC         = 10,
    SKY_CURRENT_MON_INTERVAL     = 11,
    SKY_WAIT_START_CHARGING_SEC  = 12,
};

enum sky_dev_hw_state {
    SKY_UNKNOWN                    = 0,
    SKY_SCANNING_INIT              = 1,
    SKY_SCANNING_RUN_STATE         = 2,
    SKY_SCANNING_CHECK_MATRIX      = 3,
    SKY_SCANNING_CHECK_WATER       = 5,
    SKY_SCANNING_WET               = 6,
    SKY_SCANNING_DETECTING         = 7,
    SKY_PRE_CHARGING_INIT          = 8,
    SKY_PRE_CHARGING_RUN           = 9,
    SKY_PRE_CHARGING_CHECK_MATRIX  = 10,
    SKY_PRE_CHARGING_CHECK_WATER   = 12,
```

```
    SKY_PRE_CHARGING_WET             = 13,
    SKY_PRE_CHARGING_FIND_CHARGERS   = 14,
    SKY_CHARGING_INIT                = 15,
    SKY_CHARGING_RUN                 = 16,
    SKY_CHARGING_MONITOR_CURRENT     = 17,
    SKY_POST_CHARGING_INIT           = 18,
    SKY_POST_CHARGING_RUN            = 19,
    SKY_POST_CHARGING_CHECK_MATRIX   = 20,
    SKY_POST_CHARGING_CHECK_WATER    = 22,
    SKY_POST_CHARGING_WET            = 23,
    SKY_POST_CHARGING_FIND_CHARGERS  = 24,
    SKY_OVERLOAD                     = 25,
    SKY_AUTOSCAN_DISABLED            = 250,
};
```

Skysense control commands

Commands consist of ZMQ requests and responses. In order to communicate with skyserver or skybroker client must create ZMQ socket with ZMQ_REQ type.

There are two sets of commands: one set is for accessing devices and another set is accessing remote peer (peer can be either a sky charging pad either skybroker, which acts as a transparent router).

Peer requests/responses:

Each peer request consists of only one ZMQ frame, which format is described below. Client must start communication with SKY_PEER_INFO checking protocol versions and then get list of devices from the peer, requesting SKY_DEVS_LIST:

SKY_DEVS_LIST - get devices list
Request:
  le16 type   : 0x11

Response:
  le16 type   : 0x12
  le16 errno   : POSIX.1 error number, 0x0 in case of success
  le16 num_devs   : number of devices
  struct {
    le16 dev_type   : type of the device
    le16 padding
    le32 firmware_version   : device firmware version
    unsigned char dev_uuid[16] : UUID of the device
    char portname[32]   : path to the port of a device
  } devs[num_devs]   : variable size array

SKY_PEER_INFO - get peer information
Request:
  le16 type : 0x13

Response:
  le16 type : 014
  le16 errno : POSIX.1 error number, 0x0 in case of success
  le16 proto_version : protocol version
    Major: (proto_version >> 8) & 0xff
    Minor: proto_version & 0xff
  le16 padding
  le32 server_version : server version
    Major: (server_version >> 16) & 0xff
    Minor: (server_version >> 8) & 0xff
    Revision : server_version & 0xff
  char reserved[52];

Device requests/responses:

Each device request consists of three ZMQ frame, which format is the following:
  REQ
  DEVPORT (taken from from @SKY_DEVS_LIST.devs[n].portname)
  DEVUUID (taken from from @SKY_DEVS_LIST.devs[n].dev_uuid)

Where REQ frame is a request to the device:

SKY_GET_DEV_PARAMS - receive device configuration parameters
Request:
  le16 type : 0x1
  le16 unused : reserved field
  le32 dev_params_bits : bits which represent what device params should be received,
    see enum sky_dev_param
Response:
  le16 type : 0x2
  le16 errno : POSIX.1 error number, 0x0 in case of success
  le32 dev_params[] : variable size array, represents parameters requested by
    dev_params_bits of the request

SKY_SET_DEV_PARAMS - update device configuration parameters
Request:

le16 type : 0x3
le16 unused : reserved field
le32 dev_params_bits : bits which represent what device params should be updated,
see enum sky_dev_param
le32 dev_params[] : variable size array, represents parameters to be updated

Response:
le16 type : 0x4
le16 errno : POSIX.1 error number, 0x0 in case of success

SKY_START_CHARGE - start charging
Request:
le16 type : 0x5

Response:
le16 type : 0x6
le16 errno : POSIX.1 error number, 0x0 in case of success

SKY_STOP_CHARGE - stop charging
Request:
le16 type : 0x7

Response:
le16 type : 0x8
le16 errno : error number, 0x0 in case of success

SKY_CHARGING_STATE - get charging state
Request:
le16 type : 0xd

Response:
le16 type : 0xe
le16 errno : POSIX.1 error number, 0x0 in case of success
le16 voltage : voltage in mV
le16 current : current in mA
le16 dev_hw_state : device state, see enum sky_dev_hw_state
le16 unused : reserved field

SKY_RESET_DEV - reset device
Request:
le16 type : 0xf

Response:
le16 type : 0x10
le16 errno : POSIX.1 error number, 0x0 in case of success

Skysense events

In order to receive events client must create ZMQ socket with ZMQ_SUB type and be subscribed on a device using subscription topic in the following format:

@SKY_DEVS_LIST.portname
@SKY_DEVS_LIST.dev_uuid

E.g. in C code that can be written:

```
char topic[128];
memcpy(topic, devdesc->dev_uuid,
        sizeof(devdesc->dev_uuid));
memcpy(topic + sizeof(devdesc->dev_uuid), devdesc->portname,
        strlen(devdesc->portname));
zmq_setsockopt(sub, ZMQ_SUBSCRIBE, topic,
                sizeof(devdesc->dev_uuid) + strlen(devdesc->portname));
```

Events are sent to the subscribers approximately each second.

SKY_CHARGING_STATE_EV - charging station event
le16 type : 0x80
le16 errno : POSIX.1 error number, 0x0 in case of success
le16 voltage : voltage in mV
le16 current : current in mA
le16 dev_hw_state : device state, see enum sky_dev_hw_state
le16 unused : reserved field