Shenzhen Spot Interactive Technology Co.

Smart Home Docking Protocol

4.2

catalogs

1 Basic requirements for docking

1、 Able to provide all equipment list data, equipment information to include equipment ID, equipment name, equipment type, equipment status (optional), room ID (optional), floor ID (optional).

2、 Be able to provide all scene list data, scene information to include scene ID, scene name.

3、 Ability to provide all room list data, room information to include room ID, room name. (Optional)

4、 Ability to provide all floor list data, floor information to include floor ID, floor name. (Optional)

2 Connections

1、 The background tone host sends a UDP broadcast on port (tentative) 6666 with the following data:

```
{
  "type": "REQUEST_TCP".
  //Background tone host sn code
  "sn": "xxxxx"
}
```

2、 After the server receives the broadcast sent by the background tone host, it sends a UDP broadcast on port (tentative) 7777 with the following data:

```
{
   "type": "RESPONSE_TCP".
   "data":{
          "ip": "xxx.xxx.xxx.xxx",
          "port":8888,
           "company": "xxxx"        //the company field represents the docking partner
           "protocolVersion": "v1"      //protocolVersion field indicates the version to be used (v1 and v2 are optional. The protocol format is changed accordingly. v1 protocol header is 0xAA, 2 bytes in length. v2 protocol header is 0xAB, 3 bytes in length. v2 version is able to receive more device data)
          }
```

3、 After the background sound host gets the IP of the server, it connects to the server via TCP on port (tentative) 8888.

4、 The data is then obtained via the following protocol format.

3 Protocol format

protocolVersion=v1/v2

Protocol header (0XAA)	Length (0XXXXX)	Instruction number (0XXXXX)	digital	Calibration (0XXXXX)
one byte, v1 is fixed at 0xAA, the v2 is 0xAB	v1 is two bytes and v2 is three bytes, which is the length of the "data" portion of the protocol, excluding the command number and	Two bytes, see instruction allocation rules	The content of the command, the content is a UTF-8 encoded string	A byte that is an iso of 0 with each of the bytes in "length, instruction, number, data".

checksum.

Protocol length: (v1 total protocol length = 6 + data length) (v2 total protocol length = 7 + data length)

Verification algorithm: (multiply 0 with each byte of "length, instruction number, data")

```
private boolean check(byte fcs, byte[] data) {
	int xorResult = 0;
	for (byte b : data) {
		xorResult ^= b.
	}
	return fcs == xorResult;
}
```

4 Instruction Assignment Rules

Instruction No.	sender	receiving party	Instruction type
0x0001	client	gateway	Heartbeat request
0x0002	gateway	client	heartbeat response
0x0003	client	gateway	Send control commands
0x0004	gateway	client	Responds to control commands with updates on device status
0x0005	client	gateway	Request Device List
0x0006	gateway	client	Return to the device list, which has been updated
0x0007	client	gateway	Request Scenario List
0x0008	gateway	client	Back to the list of scenarios
0x0009	client	gateway	Request room list (optional)
0x000A	gateway	client	Return to room list (optional)
0x000B	client	gateway	Request floor list (optional)
0x000C	gateway	client	Return to floor list (optional)
0x000D	client	gateway	(Reserved)
0x000E	gateway	client	Let the background sound host speak

0x000F	gateway	client	actively (optional) Recapture device scene data on the notification screen side

Note: Client refers to the background sound host

5 Data sent by each command

5.1 Heartbeat Request

```
{
   "type": "REQUEST_HEART_BEAT".
   "sn": "xxxxx"
}
```

5.2 Heartbeat Response

```
{
   "type": "RESPONSE_HEART_BEAT"
}
```

5.3 Requesting a list of devices

```
{
   "type": "REQUEST_DEVICE ",
   "sn": "xxxxx"
}
```

5.4 Returning to the device list with updates to the device list

```
{
   "type": "RESPONSE_DEVICE"
   "deviceData":[
         // Specific return data
         {
         "deviceId": "deviceId",
         "deviceName": "deviceName",
         "deviceType": "deviceType".
         "roomId": "roomId (optional)",
         "floorId": "floorId (optional)",
         "state":{
                     "order": "Device Command On, Off, Pause
         (optional)",.
                     "mode": "Device mode (optional)",.
                     "temperature": "Temperature (optional)".
                     "fanSpeed": "windSpeed (optional)".
                     "fanDirection": "windDirection (optional)".
                     "brightness": "Brightness 0-100 (optional)",
                     "position": "Progress 0-100 (optional)",
                      "angle": "Angle 0-180".
                      "sensorStatus": "1: occupied, alarm triggered 0:
unoccupied, no alarm triggered",
                      "battery": "Battery".
                      "humidity": "humidity".
                      "universalSensorData":
"universalSensorData(jsonArray.toJSONString())"
                  }
         }, }
         {
         "deviceId": "deviceId",
         "deviceName": "deviceName",
         "deviceType": "deviceType".
         "roomId": "roomId (optional)",
         "floorId": "floorId (optional)",
         "state":{
                     "order": "Device    On, Off, Pause (optional)",.
                     "mode": "Device mode (optional)",.
                     "temperature": "Temperature (optional)".
                     "fanSpeed": "windSpeed (optional)".
                     "fanDirection": "windDirection (optional)".
```

```
            "brightness": "Brightness 0-100 (optional)",
            "colorTemp": "Color temperature value 0-100 (optional)",.
            "position": "Progress 0-100 (optional)",
            "angle": "Angle 0-180".
            "sensorStatus": "1: occupied, alarm triggered 0: unoccupied, no alarm
triggered",
            "battery": "Battery".
            "humidity": "humidity".
            "universalSensorData":
"universalSensorData(jsonArray.toJSONString())"
        }
    }
]
```

universalSensorData Description

```
Universal Sensor type=MULTIFUNCTIONAL_SENSOR
universalSensorData=jsonArray.toJSONString(); specific sensor data, the value
        is in the form of a jsonArray and can be passed as follows
        JSONArray jsonArray = new JSONArray();
        jsonArray.add(JSONObject.parse("{'key':'key1xxx','value':'value1xxx'}"));
        jsonArray.add(JSONObject.parse("{'key':'key2xxx','value':'value2xxx'}"));
        jsonArray.add(JSONObject.parse("{'key':'key3xxx','value':'value3xxx'}"));
        jsonArray.add(JSONObject.parse("{'key':'key4xxx','value':'value4xxx'}"));
You can set the one you want to display by yourself, key means name, for
example, jsonArray.add(JSONObject.parse("{'key':'illumination
value','value':'50Lux'}")");
```

5.5 List of request scenarios

```
{
   "type": "REQUEST_SCENE".
   "sn": "xxxxx"
}
```

5.6 Returning to the Scene List

```
{
   "type": "RESPONSE_SCENE".
   "data":[
        // Specific returned data
        {
           "id": "Scene ID".
           "name": "Scene name"
        }, }
        {
           "id": "Scene ID".
           "name": "Scene name"
        }
    ]
}
```

5.7 Request for room list (optional)

```
{
   "type": "REQUEST_ROOM".
   "sn": "xxxxx"
}
```

5.8 Return to room list (optional)

```
{
   "type": "RESPONSE_ROOM".
   "data":[
          // Specific return data
          {
              "id": "roomid",
              "name": "Room name"
           }, }

              "id": "roomid",
              "name": "Room name"
          }
    ]
}
```

5.9 Request floor list (optional)

```
{
   "type": "REQUEST_FLOOR".
   "sn": "xxxxx"
}
```

5.10 Return to floor list (optional)

```
{
   "type": "RESPONSE_FLOOR".
   "data":[
        // Specific return
        {
              "id": "FloorID",
              "name": "Floor name"
        }, }
        {
               "id": "FloorID",
              "name": "Floor name"
        }
    ]
}
```

5.11 Sending control commands

```
{
   "type": "REQUEST_CONTROL".
   "sn": "xxxxx".
   "original": "original language".
   "data":[
          {
           //Specific control commands to be sent
           "id": "device-id".
           "name": "Device Name",
           "scene": "scene".
           "state": "status".
           "action": "Action".
           "attribute": "attributes".
           "attributeValue": "attributeValue".
           "mode": "mode".
           "floor": "Floor".
           "room": "room"
          }, }
          {
           //Specific control commands to be sent
           "id": "device-id".
           "name": "Device Name",
           "scene": "scene".
           "state": "status".
           "action": "Action".
           "attribute": "attributes".
           "attributeValue": "attributeValue".
           "mode": "mode".
           "floor": "Floor".
           "room": "room"
          }
    ]
}
```

5.12 Responding to control commands, or if there is an update to the device status

```
{
    "type": "RESPONSE_CONTROL".
    //return code
     "code":0
    "msg": "Return message",
    // is empty then the default answer is [OK].
     "msgVoice": "Requires background voice host to say",
     "deviceId": "deviceId",
     "deviceName": "deviceName",
     "deviceType": "deviceType".
      "deviceState":{
            //Specifically return device status data
            "order": "Device commands on, off, pause".
            "mode": "Device Mode".
            "temperature": "Temperature".
            "fanSpeed": "windSpeed".
            "fanDirection": "windDirection".
            "brightness": "Brightness 0-100".
            "position": "Progress 0-100 ".
            "angle": "Angle 0-180".
             "sensorStatus": "1:occupied, triggered 0:unoccupied, not
triggered",
             "battery": "Battery".
             "humidity": "humidity".
             "universalSensorData":
"universalSensor(jsonArray.toJSONString())"
               }
}
```

Note: When the order field is empty in 5.4 Getting Device List and 5.12 Status Refresh, it will be set to STATE_OFF|Off by default.

5.13 Let the background sound host, active speech (optional)

```
{
   "type": "RESPONSE_VOICE".
   //What needs to be said actively by the background sound host
   "msgVoice": "Need background voice host to initiate words"
}
```

6 Return Status Code

status code	The meaning of the expression
0	Control success
1	This device does not exist
2	The device does not support this operation
3	Control incoming parameters are incorrect
4	Equipment offline
5	Device control timeout
6	(sth. or sb) else

7 Common Device Types

device name	English Type
lantern	LIGHT
dimmer	DIMMING_LAMP
sockets	SOCKET
window curtains	CURTAIN
Adjustable Progress Curtains	ADJUST_CURTAIN
fan (loanword)	ELECTRIC_FAN
refrigeration	AIR_CONDITION
radio	TV
thermostat	THERMOSTAT
projectors	PROJECT
illusionary curtains	CURTAIN_ANGLE
door and window magnet	DOOR_MAGNETIC_SENSOR
rainstorm	DOOR_WATER_SENSOR
human infrared (HRI)	INFRARED_SENSOR
fumes	SMOKER_SENSOR
SOS	SOS_SENSOR
Universal Sensor	DEVICE_TYPE_MULTIFUNCTIONAL_SENSOR
Temperature and humidity display	TEMP_HUMIDITY_SENSOR

new custom	FRESH_AIR
floor heating	FLOOR_HEATING
switchgear	SWITCH
color temperature lamp	CW_LAMP

8 Control commands

```
{
   "id": "device-id".
   "name": "Device Name",
   "scene": "scene".
   "state": "status".
   "action": "Action".
   "attribute": "attributes".
   "attributeValue": "attributeValue".
   "mode": "mode".
   "floor": "Floor".
   "room": "room"
}
```

Fantasy Curtain (with angle adjustment).

Control down message: state= STATE_ON (on, progress 100), state=STATE_OFF (off, progress 0), state=STATE_STOP (suspended, progress 50)

Progress: action=ACTION_TO, attribute=ATTRIBUTE_PROGRESS, attributeValue=specific progress value

Angle: action=ACTION_TO, attribute=ATTRIBUTE_ANGLE, attributeValue=specific angle value

(Note: the first English or numeric number to the left of the | sign is the data value in the json, the rest of the | sign is the meaning of the English or numeric representation)

- **state of affairs**

  STATE_ON|On

  STATE_OFF|Off

  STATE_STOP|Stop

- **movements**

ACTION_TO|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To|To

ACTION_ADD|Brighten|Higher|Warmer|Hotter|Bigger|Increase|Bigger|Increase |Brighten |Brighten |Higher|Higher |Next |Raise |Bigger |Bigger |Greater |Increase|Increase |Raise |Pull Up |Pull Apart |Pull Up |Pull Up

ACTION_REDUCE|Darken|Lower|Cooler|Smaller|Decrease|Reduce|Darker|Darker|Lower|Lower|Lower|Previous|Decrease|Smaller|Smaller|Decrease|Pull Down|Pull Down

- **paradigm**

  MODE_COOL|Cooling

  MODE_HEAT|Heating

  MODE_FAN|Air Supply

  MODE_DRY|Dehumidifying|Drying

  MODE_AUTO|Automatic

  MODE_WIND_HIGH|HIGH WIND

  MODE_WIND_MID|Stroke

  MODE_WIND_LOW|Low Wind

  MODE_WIND_AUTO|Automatic wind

  MODE_WIND_MUTE|Mute Wind

  MODE_PLAY|Playback

  MODE_PAUSE|Pause

  MODE_MUTE|Mute

  MODE_NEXT|Next song|Air conditioner mode switching

MODE_PREVIOUS|Previous Article

MODE_SWING_ANGLE|Broadcasting Heads|Pendulum Winds

MODE_SWING_NO|Stop Swinging Wind|Stop Swinging Head

MODE_SWING_LEFT_RIGHT|Left/Right Wind Swing

MODE_SWING_UP_DOWN|Up and Down Winds

MODE_PURIFIER|Air Purification

- **causality**

ATTRIBUTE_MODE|Mode

ATTRIBUTE_WIND_SPEED|Wind

ATTRIBUTE_WIND_DIRECTION|Wind Direction

ATTRIBUTE_TEMPERATURE|Temperature

ATTRIBUTE_GEAR|Gears

ATTRIBUTE_BRIGHTNESS|Brightness

ATTRIBUTE_COLOR|Color

ATTRIBUTE_COLOR_TEMP|Color

ATTRIBUTE_VOLUME|Volume| Sound

ATTRIBUTE_CHANNEL|Channel| Program|

ATTRIBUTE_HOUR|hours

ATTRIBUTE_MINUTE|minutes

ATTRIBUTE_SECOND|seconds

ATTRIBUTE_PROGRESS| Progress

ATTRIBUTE_ANGLE| Angle

- **attribute value**

VALUE_COLOR_WHITE|White

VALUE_COLOR_RED|Red

VALUE_COLOR_ORANGE|Orange

VALUE_COLOR_BLUE |

VALUE_COLOR_PINK|Pink| pink| pink

VALUE_MAX|Brightest| Maximum| Highest|

VALUE_MIN|Darkest| Smallest| Lowest|

VALUE_LITTLE|one point| one point

0.5|half| one-half

1.5|one and a half

2.5|two and a half| two and a half

3.5|three

4.5|four and a half