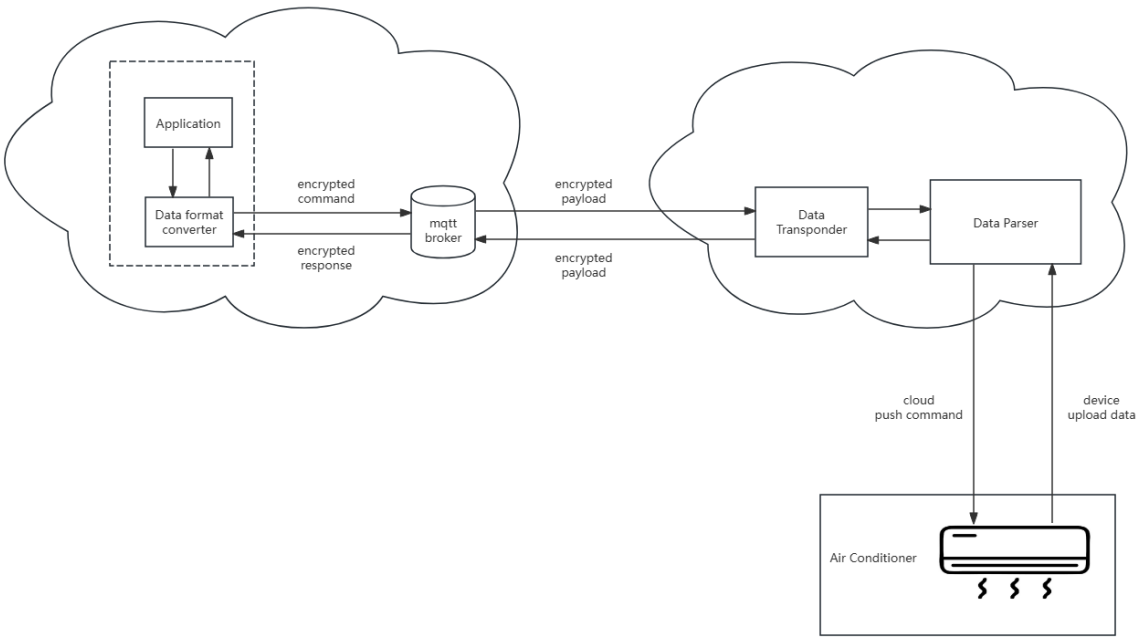
MQTT Communication Protocol For Air Conditioner

Version	Date	Editor	Content
V1.0	2025-06-03	Hanleo	- **init**

1. Architecture diagram



The device (Air-Conditioner) automatically reports device data to the cloud based on the reporting policy. Device may periodically report all data or report only the changed device status data, which will be parsed into key-value formated by Data Parser. These parsed data will be encrypted

and sent to mqtt broker. For commands, client-side needs to publish the encrypted command to mqtt broker. RAC is responsible for encoding and pushing the command to devices.All in all, client-side needs to develop a data adapter according to the protocol below.

2. Encryption

Data encryption refers to encrypting the communication messages between RAC and client-side. The encrypted content includes device status data from RAC to client-side and command data from client-side to Rac. The encryption method used is RSA encryption. The specific encryption process for the messages is as follows:

1. Serialize the message into a JSON string.
2. Encrypt the JSON string with public key to obtain an encrypted byte array.
3. Base64 encode the RSA-encrypted message to obtain the final message.
   The data decryption process is the reverse of the above steps, and can be decrypted using the corresponding private key. The public key is determined and provided by each platform. Specific encryption and decryption process is shown below via Java.

Sample

1. Generate key pairs

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Base64;

public class Demo {
    public static void main(String[] args) throws Exception {

        // 1. create KeyPairGenerator using RSA
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");

        // 2. initialize KeyPairGenerator with key length of 2048
        keyPairGenerator.initialize(2048);

        // 3. generate KeyPair
        KeyPair keyPair = keyPairGenerator.generateKeyPair();

        // 4. get PublicKey and privateKey
        PublicKey publicKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        // 5.  Convert PublicKey and PrivateKey objects to Base64 encoded
strings
        String publicKeyString =
Base64.getEncoder().encodeToString(publicKey.getEncoded());
        String privateKeyString =
Base64.getEncoder().encodeToString(privateKey.getEncoded());

        // 6. output private key string and public key string
        System.out.println("Public Key: " + publicKeyString);
        System.out.println("Private Key: " + privateKeyString);
    }
}
```

After executing codes above, we have the private key and public key. Keys below are for demonstration use only.

```
Public Key:
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAydunx5IvJriP88xko1UpgOfilQWfxS9Qxpu8
E96K71Nwva8yxO7oLDSd+EXmPULX1Na5FnKmSp8YerKz8+AqOKt+4vUBK/1xqUWGK7QxPW13kufI+ogw
GBw2BuGhi5AeOKOX6mZyWud5tYkY8QRWXu/axrJYmw2ARxp+jEJ98r9SwK4X9Y6m2oEzjg3n+X5fjnKu
ppO6TbcsoJd9SCJEf1zfv/RCqEOj9HyZEDHHtqn3Ej89djQbElKADHO6iGDB9V2y9KL2ZBALeh5382Z6
TKBr7v+T45CMwlB/h/smcODjTgoSk2NsOWJNdmAxIdezV9C9K3LLcQPh4eQuIsAWfwIDAQABPrivate
Key:MIIEwAIBADANBgkqhkiG9w0BAQEFAASCBKowggSmAgEAAoIBAQDJ26fHki8muI/zzGSjVSmA5+KV
BZ/FL1DGm7wT3orvU3C9rzLE7ugsNJ34ReY9QtfU1rkWcqZKnxh6srPz4CrQq37i9QEr/XGpRYYrtDE9
bXeS58j6iDAYHDYG4aGLkB7QrRfqZnJa53m1iRjxBFZe79rGslibDYBHGn6MQn3yv1LArhf1jqbagTOO
Def5fl+Ocq6mnTpNtyygl31IIkR/XN+/9EKoQ6POfJkQMce2qfcSPz12NBsSUoAMfTqIYMH1XbL0ovZk
EAt6HnfzZnpMoGvu/5PjkIzCUH+H+yZzQONOChKTY2zRYk12YDEh17NXOL0rcstxA+Hh5C4iwBZ/AgMB
AAECggEBAKzRt1KAOIeZKf2zt1GzPsZTQUUOvJwIh2KSwLJOLKr7yCFOFuxOEk6OedSum5FFYh2h/HLA
k6h9j2q1BqY7/MwMl6Td2DP+V8pxcI6IRkPuLZqDgSklYe+yb3ryv+kJ6agFGd2f+jXjyernRohKi1va
hHbrCDsvkuzNPRz+bHBfX5BOiQ1G6xJpB4Ugq1KDsE2UP/I9ObW+8ebYgK34vNrUl1ObWomLypLzq2NF
7svoQJkUDnItd1HmsEyZdQizYOfPgvZ1oDnwFtStF/zCRIO+sCM+s1pykSdX++8A36x1Ug/+ZxDXKgBv
rP9Xb92hwCOW1UntgplnZf9ozJmB1JkCgYEA/Q1zeG1YbOB8AHE68BJQYy4f184zdRND+2tceR54jUSP
M938Rdt57bO8ShX/u7ldPJOzxhL2mv2b+sW52IAHGxWvmKdoNJyAihtI8WvXcUZIHnLS7zSl1h39TOpJ
tcDpoHwOiSvpH+SeJgLaMQE6msIOooax7qNdtiJmsyNvru0CgYEAzDWNnBaU6B7c7SZXOIyH58Zvedg+
0/QiMLrVcm2hG2SuVcqDnCSQtOQxrMPsKqHbXpo8VoXGSazn5UVnmY7gAvVzXr6dgpqatkdg3Nla3Hwx
OptYsSSV175btJiTTS9QSqgcsHxjCD8k/gFHolNegy88O+2umviAJRCToa+zQZsCgYEAyVjUkfPfM9ZJ
VEjELpVD5UakC6CEjRkhNIG+eJjoLu9oSkrNGgCCScoSmJhcfVAGgd5LfcVE9xnn7Gc7cqqb/M8Kgxpg
7cvA8yK2rXGzYOwdIiOW7UIk2fPOxDJF6BpxSkLJwoNv4PKVa4VFeOyRBuaT3uXx2gGe5PcGZG9DxBkC
gYEAl2WsO2g6hcO75eZjC93EdKW9yAv2qJV7M72QugFlTDXcumYmJQFG+Vhpq7ilwwwO06fJXgZ4OYDL8
yhImuMYgdpEbM8cGZ+nPggxxV8sdfhHlrkrV+p8xKH5M5xunObY2RONWQdI2acrG1jqErWg5zfCbK6MP
rH1IJDKiAWGN+LkCgYEA2YaW7CHeOHeNhMiGnsDxhuzVEHRwNiNOUKh9jheiqG4yoG9G5Wd2G8dtb+qr
7OlwraM6b9QOOGu7rDGgJLfeeHAbb1jL72HM5TA2MCiTth+XOXUWI6qnuvNuDKDEu9tE0bfMx1cP1zhG
wJtPdAOY/Q791SRKvl/TQRTT/2p1mSM=
```

2.Encryptmessagewithpublickey

```
import javax.crypto.Cipher;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

public class Demo {
    public static void main(String[] args) throws Exception {
        String messageToSend = "{\"name\":\"Alice\",\"age\":20}";
        String publicKeyString =

 "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAydunx5IvJriP88xko1UpgOfilQWfxS9Qxp
u8E96K71Nwva8yxO7oLDSd+EXmPULX1Na5FnKmSp8YerKz8+AqOKt+4vUBK/1xqUWGK7QxPW13kufI+o
gwGBw2BuGhi5AeO
KOX6mZyWud5tYkY8QRWXu/axrJYmw2ARxp+jEJ98r9SwK4X9Y6m2oEzjg3n+X5fjnKuppO6TbcsoJd9S
CJEf1zf
v/RCqEOj9HyZEDHHtqn3Ej89djQbElKADHO6iGDB9V2y9KL2ZBALeh5382Z6TKBr7v+T45CMwlB/h/sm
cODjTgo Sk2NsOWJNdmAxIdezV9C9K3LLcQPh4eQuIsAWfwIDAQAB";

        // generate public key object from public key string
        byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyString);
        X509EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(publicKeyBytes);
```

```
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);

        // encrypt message with public key
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(messageToSend.getBytes());
        String encryptedMessage =
Base64.getEncoder().encodeToString(encryptedBytes);
        System.out.println("Encrypted message: " + encryptedMessage);
    }
}
```

Output

```
Encrypted message:
EXgQcjt6ITjeGRfmFDBHZk1f+OaiNQuCN4bxNFpKsGO4s1fF47fFZ5ndkKX0aH7pMrUVx0D6zu/4mcDE
oGtA9isBElZLRsMLFmKmRP2TxZCTgOBYQVxeSJ6j+R4jB729CWOwiGC+Rj1D99sLgqkFu5tsOGPLda1G
KyBFTS/9iEmT2TNf4TqtsussGY0Vro76xtODiqNiewuPhvFHBsiivK+gwAwZWLbKz2FSzZ8N7j8kEA6C
fQkVfxG/ANa8/DB66QDFpHwajlTctvJ1sKM/BS3djvHpfviB7kwIYHcnd2jOgskUee+nimVSbRkkyvWl
uWOPes+nVtZgB3vqRJKKyg==
```

3.Decrypt message with private key

```
package com.midea.orac.oemopen.api;


import javax.crypto.Cipher;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.Base64;

public class Demo {
    public static void main(String[] args) throws Exception {
        String messageReceived =
"EXgQcjt6ITjeGRfmFDBHZk1f+OaiNQuCN4bxNFpKsGO4s1fF47fFZ5ndkKX0aH7pMrUVx0D6zu/4mcD
EoGtA9isBElZLRsMLFmKmRP2TxZCTgOBYQVxeSJ6j+R4jB729CWOwiGC+Rj1D99sLgqkFu5tsOGPLda1
GKyBFTS/9iEmT2TNf4TqtsussGY0Vro76xtODiqNiewuPhvFHBsiivK+gwAwZWLbKz2FSzZ8N7j8kEA6
CfQkVfxG/ANa8/DB66QDFpHwajlTctvJ1sKM/BS3djvHpfviB7kwIYHcnd2jOgskUee+nimVSbRkkyvW
luWOPes+nVtZgB3vqRJKKyg==";
```

```
        String privateKeyString =
 "MIIEwAIBADANBgkqhkiG9w0BAQEFAASCBKowggSmAgEAAoIBAQDJ26fHki8muI/zzGSjVSmA5+KVBZ
/FL1DGm7wT3orvU3C9rzLE7ugsNJ34ReY9QtfU1rkWcqZKnxh6srPz4CrQq37i9QEr/XGpRYYrtDE9bX
eS58j6iDAYHDYG4aGLkB7QrRfqZnJa53m1iRjxBFZe79rGslibDYBHGn6MQn3yv1LArhf1jqbagTOODe
f5fl+Ocq6mnTpNtyygl31IIkR/XN+/9EKoQ6POfJkQMce2qfcSPz12NBsSUoAMfTqIYMH1XbL0ovZkEA
t6HnfzZnpMoGvu/5PjkIzCUH+H+yZzQONOChKTY2zRYk12YDEh17NXOL0rcstxA+Hh5C4iwBZ/AgMBAA
ECggEBAKzRt1KAOIeZKf2zt1GzPsZTQUUOvJwIh2KSwLJOLKr7yCFOFuxOEk6OedSum5FFYh2h/HLAk6
h9j2q1BqY7/MwMl6Td2DP+V8pxcI6IRkPuLZqDgSklYe+yb3ryv+kJ6agFGd2f+jXjyernRohKi1vahH
brCDsvkuzNPRz+bHBfX5BOiQ1G6xJpB4Ugq1KDsE2UP/I9ObW+8ebYgK34vNrUl1ObWomLypLzq2NF7s
voQJkUDnItd1HmsEyZdQizYOfPgvZ1oDnwFtStF/zCRIO+sCM+s1pykSdX++8A36x1Ug/+ZxDXKgBvrP
9Xb92hwCOW1UntgplnZf9ozJmB1JkCgYEA/Q1zeG1YbOB8AHE68BJQYy4f184zdRND+2tceR54jUSPM9
38Rdt57bO8ShX/u7ldPJOzxhL2mv2b+sW52IAHGxWvmKdoNJyAihtI8WvXcUZIHnLS7zSl1h39TOpJtc
DpoHwOiSvpH+SeJgLaMQE6msIOooax7qNdtiJmsyNvruOCgYEAzDWNnBaU6B7c7SZXOIyH58Zvedg+O/
QiMLrVcm2hG2SuVcqDnCSQtOQxrMPsKqHbXpo8VoXGSazn5UVnmY7gAvVzXr6dgpqatkdg3Nla3HwxOp
tYsSSV175btJiTTS9QSqgcsHxjCD8k/gFHolNegy88O+2umviAJRCToa+zQZsCgYEAyVjUkfPfM9ZJVE
jELpVD5UakC6CEjRkhNIG+eJjoLu9oSkrNGgCCScoSmJhcfVAGgd5LfcVE9xnn7Gc7cqqb/M8Kgxpg7c
vA8yK2rXGzYOwdIiOW7UIk2fPOxDJF6BpxSkLJwoNv4PKVa4VFeOyRBuaT3uXx2gGe5PcGZG9DxBkCgY
EAl2WsO2g6hcO75eZjC93EdKW9yAv2qJV7M72QugFlTDXcumYmJQFG+Vhpq7ilwwwO6fJXgZ4OYDL8yh
ImuMYgdpEbM8cGZ+nPggxxV8sdfhHlrkrV+p8xKH5M5xunObY2RONWQdI2acrG1jqErWg5zfCbK6MPrH
1IJDKiAWGN+LkCgYEA2YaW7CHeOHeNhMiGnsDxhuzVEHRwNiNOUKh9jheiqG4yoG9G5Wd2G8dtb+qr70
lwraM6b9QOOGu7rDGgJLfeeHAbb1jL72HM5TA2MCiTth+XOXUWI6qnuvNuDKDEu9tE0bfMx1cP1zhGwJ
tPdAOY/Q791SRKvl/TQRTT/2p1mSM=";
        // generate private key object from private key string
        byte[] privateKeyBytes = Base64.getDecoder().decode(privateKeyString);
        PKCS8EncodedKeySpec privateKeySpec = new
PKCS8EncodedKeySpec(privateKeyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PrivateKey privateKey = keyFactory.generatePrivate(privateKeySpec);
        // decrypt with private key
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes =
cipher.doFinal(Base64.getDecoder().decode(messageReceived));
        String decryptedMessage = new String(decryptedBytes);
        System.out.println("Decrypted message: " + decryptedMessage);
    }
}
```

Output

```
Decrypted message: {"name":"Alice","age":20}
```

3. Device Status Data Forward

3.1 Device Status Data Forward

Device status data refers to the key-values generated during the device operation. Device status data mentioned in this document refers to the runtime data of the device.
The RAC is responsible for parsing the raw data reported by the device and sending the parsed data to the client-side.
The reporting period is determined by the device.
After the data is reported, it will be immediately forwarded to the client-side after parsing, without any additional delay except for data structure conversion.
The points that need to be forwarded are determined by the client-side after screening.
If not screened, RAC will transmit all parsed data reported by the device to client-side in full.

Message Information

Protocol	MQTT
OperationType	DEV_RT_REPORT
Usage	The RAC forwards reported data from device to client-side. Client-side does not need to reply.
Topic	md/dev/report/{deviceId}

Message Structure

Field	Required	Type	Description
op	Y	String	Operation Type(constant value: DEV_RT_REPORT)
seqNo	Y	Integer	Sequence number of message
deviceType	Y	String	Device type provided by RAC. ex. AC_2530001N
payload	Y	String	JSON-formatted string of structured data below.

Payload Structure

Field	Required	Type	Description
timestamp	Y	String	Data reporting time (timestamp, accurate to milliseconds).
deviceInfo	Y	Object	Device information. It's structure can be seen below.
data	Y	Object[]	The parsed data of reported device point and value. It's structure can be seen below.

DeviceInfo Structure

Field	Required	Type	Description
deviceId	Y	String	The unique code assigned to devices By RAC
sn	Y	String	Appliance unique SN, one-to-one correspondence with deviceId

Data Structure

Field	Required	Type	Description
name	Y	String	The name of parsed key.
value	Y	String	The pared value from reportd data.
unit	Optional	String	Unit of the reported data. For example, the unit of temperation can be Celsius/Fahrenheit.

Complete definition in "Device Status Definitions"

Sample


```
{
  "op": "DEV_RT_REPORT",
  "seqNo": 1,
  "deviceType": "CD_2530001N",
  "payload": "{\"timeStamp\":\"1728528545742\",\"deviceInfo\":
{\"deviceId\":\"123456\",\"sn\":\"ABC1111222233333\"},\"data\":
[{\"name\":\"power\",\"value\":\"on\"},{\"name\":\"mode\",\"value\":\"energy\"},
{\"name\":\"set_temperature\",\"value\":\"37\",\"nuit\":\"Celsius\"},
{\"name\":\"vacation\",\"value\":\"off\"},
{\"name\":\"set_vacationdays\",\"value\":\"0\"},
{\"name\":\"water_box_temperature\",\"value\":\"30\",\"nuit\":\"Celsius\"},
{\"name\":\"outdoor_temperature\",\"value\":\"15\",\"nuit\":\"Celsius\"}]}"
}
```


4. Device Status Query Command

4.1 Device Status Query Command

Client-side retrieves the device status data stored by RAC with the device status query command, which may have a certain delay compared to the on-site device status. The specific delay depends on the reporting strategy of the on-site device.

Message Information

Protocol	MQTT
Operation Type	DEV_GET_DATA
Usage	Client-side proactively retrieves the status data of a specific device from the RAC.
Topic	md/dev/get/{deviceId}

Message Structure

Field	Required	Type	Description
op	Y	String	Operation Type(constant value: DEV_GET_DATA)
seqNo	Y	Integer	Sequence no of message.
deviceType	Y	String	Device type provided by RAC. ex. CD_2530001N
payload	Y	String	JSON-formatted string of structured data below.

SeqNo: Sequence no of message.
To match requests and responses, RAC fills in the sequence number in the response topic to match the commands one by one.
Therefore, when client-side sends a command, it needs to ensure that the seqNo of the same deviceId is unique within a certain time period (>1min).

Payload Structure

Field	Required	Type	Description
timestamp	Y	String	Data reporting time (timestamp, accurate to milliseconds).
data	Y	Object[]	The parsed data of reported device point and value. It's structure can be seen below.

Data Structure

Field	Required	Type	Description
deviceId	Y	String	The unique id of device from RAC.
dataType	N	String	The type of data. Default value: BASE;

Sample

```
{
  "op": "DEV_GET_DATA",
  "seqNo": 1,
  "deviceType": "CD_2530001N",
  "payload": "{\"timestamp\":\"1728528545742\",\"data\":
{\"deviceId\":\"178120888888888\",\"dataType\":\"BASE\"}}"
}
```

4.2 Response Message Information

Protocol	MQTT
Operation Type	DEV_GET_DATA_RES
Usage	RAC replies the status data of the corresponding device to client-side.
Topic	md/dev/get/response/{deviceId}/{seqNo}

Message Structure

Field	Required	Type	Description
op	Y	String	Operation Type(constant value: DEV_GET_DATA_RES)
SeqNo	Y	Integer	Sequence no of message, equals to the sequence number of the query command.
payload	Y	String	JSON-formatted string of structured data below.

Payload Structure

Field	Required	Type	Description
result	Y	Integer	Execute result: 0 for success, -1 for fail
errCode	N	Integer	0 for success, others for errorCodes
errMsg	N	String	Error message
timestamp	Y	String	Data reporting time (timestamp, accurate to milliseconds).
deviceInfo	Y	Object	Device information. Detailed structure can be seen in Figure 1: Device Information.
data	Y	Object[]	The parsed data of reported device point and value. It's structure can be seen below.

DeviceInfo Structure

Field	Required	Type	Description
deviceId	Y	String	The unique code assigned to devices By RAC
sn	Y	String	Appliance unique SN, one-to-one correspondence with deviceId

Data Structure

Field	Required	Type	Description
name	Y	String	The name of parsed key.
value	Y	String	The pared value from reportd data.
unit	Optional	String	Unit of the reported data. For example, the unit of temperation can be Celsius/Fahrenheit.

Sample

```
{
  "op": "DEV_GET_DATA_RES",
  "seqNo": 1,
  "payload": "{\"result\":\"0\",\"timeStamp\":\"1728528545742\",\"deviceInfo\":
{\"deviceId\":\"1222111121\",\"sn\":\"ABC111122223333\"},\"data\":
[{\"name\":\"power\",\"value\":\"on\"},{\"name\":\"mode\",\"value\":\"energy\"},
{\"name\":\"set_temperature\",\"value\":\"37\",\"nuit\":\"Celsius\"},
{\"name\":\"vacation\",\"value\":\"off\"},
{\"name\":\"set_vacationdays\",\"value\":\"0\"},
{\"name\":\"water_box_temperature\",\"value\":\"30\",\"nuit\":\"Celsius\"},
{\"name\":\"outdoor_temperature\",\"value\":\"15\",\"nuit\":\"Celsius\"}]}"
}
```


5. Device Online/Offline Events Forward

5.1 Device Online/Offline Events Forward

RAC sends this event message to client-side once the device online/offline status changes.

Message Information

Protocol	MQTT
Operation Type	DEV_EVENT_REPORT
Usage	RAC sends the information of device online and offline status change event to client-side. There's no need to reply.
Topic	md/dev/report/{deviceId}

Message Structure

Field	Required	Type	Description
op	Y	String	Operation Type(constant value: DEV_EVENT_REPORT)
seqNo	Y	Integer	Sequence no of message.
deviceType	Y	String	Device type provided by RAC ex. CD_2530001N
payload	Y	String	JSON-formatted string of structured data below.

Payload Structure

Field	Required	Type	Description
timestamp	Y	String	Data reporting time (timestamp, accurate to milliseconds).
data	Y	Object[]	The event data. It's structure can be seen below.

Data Structure

Field	Required	Type	Description
evtCode	Y	String	Event Code
evtType	Y	String	Event Type(constant value: EVENT)
deviceId	Y	String	The unique id of device from RAC.
sn	Y	String	device serial number
status	Y	String	Event status (enumerate values: offline, online), which means the specific device has changed It's status to online/offline

Sample


```
{
  "op": "DEV_EVENT_REPORT",
  "seqNo": 1,
  "deviceType": "CD_2530001N",
  "payload": "{\"timestamp\":\"1696734964660\",\"data\":
{\"evtCode\":\"EVT001\",\"evtType\":\"EVENT\",\"deviceId\":\"111111222222\",\"sn
\":\"000000P0000000Q1102C8D5E5ABC0000\",\"status\":\"offline\"}}"
}
```


6. Device Control Command

6.1 Device Control Command

client-side uses this commad to control the Air-Conditioner. Specific control command definitions will be given by an extra documention.

Only the command structure will be explained here. Timeout period of this command can be set to 30 seconds or more.

We will query the status first, and put the target data to appliance current status, then send them to appliance.
For example, when we got action equals BASE with power is on, we will query BASE, then update the status with power=on, then send the modified status to the appliance.

Message Information

Protocol	MQTT
Operation Type	DEV_CONTROL
Usage	client-side controls device via this command
Topic	md/dev/set/{deviceId}

Message Structure

Field	Required	Type	Description
op	Y	String	Operation Type(constant value: DEV_CONTROL)
seqNo	Y	Integer	Sequence number of message
deviceType	Y	String	Device type provided by RAC ex. CD_2530001N
payload	Y	String	JSON-formatted string of structured data below.

Payload Structure

Field	Required	Type	Description
timeStamp	Y	String	Data reporting time (timestamp, accurate to milliseconds).
data	Y	Object[]	The parsed data of reported device point and value. It's structure can be seen below.

Data Structure

Field	Required	Type	Description
deviceId	Y	String	the unique id of the device from RAC
action	Y	String	This is a constant value "BASE";
properties	Y	Object[]	The properties need to be set. Structure is shown below.

Properties Structure

Field	Required	Type	Description
name	Y	String	name of property.
value	Y	String	value to be set.

Sample

```
{
  "op": "DEV_CONTROL",
  "seqNo": 1,
  "deviceType": "CD_2530001N",
  "payload": "{\"timestamp\":\"1696747810722\",\"data\":
{\"deviceId\":\"178120883707921\",\"action\":\"BASE\",\"properties\":
[{\"name\":\"a\",\"value\":\"1\"}]}}"
}
```

6.2 Response Message Information

Protocol	MQTT
Operation Type	DEV_CONTROL_RES
Usage	RAC replies the control data of the corresponding device to client-side.
Topic	md/dev/set/response/{deviceId}/{seqNo}

Message Structure

Field	Required	Type	Description
op	Y	String	Operation Type(constant value: DEV_CONTROL_RES)
seqNo	Y	Integer	Sequence number of message, equals to the sequence number of the control command.
payload	Y	String	JSON-formatted string of structured data below.

Payload Structure

Field	Required	Type	Description
result	Y	Integer	Execute result: 0 for success, -1 for fail
timestamp	Y	String	Data reporting time (timestamp, accurate to milliseconds).
errCode	N	Integer	Execute result: 0 for success, others for errors
errMsg	N	String	Error message

Sample

success:

```
{
  "op": "DEV_CONTROL_RES",
  "seqNo": 1,
  "payload": "{\"result\":0,\"timestamp\":\"1728528545742\"}"
}
```

failed:

```
{
  "op": "DEV_CONTROL_RES",
  "seqNo": 1,
  "deviceType": "CD_2530001N",
  "payload": "
{\"result\":-1,\"timestamp\":\"1728528545742\",\"errCode\":\"10001\",\"errMsg\":
\"invalidCommand\"}"
}
```