IoTtalk Documentation

Release 2.0.1

IoTtalk contributors

Jul 26, 2017

Contents

CHAPTER 1

Architecture

Components

Input Unit

The input unit can be a physical device, program, or any thing that can generate data.

IoTtalk Data Gateway

The IoTtalk Data Gateway will transfer data according to user configured project.

Output Unit

The output unit can be a physical device, program, or any thing that can accept data.

CHAPTER 2

Terminology

CHAPTER 3

Protocol

Component Discovery Protocol

In a local area network, we may have one or more input/output devices. We will introduce how to plug/remove devices here.

state Draft

Preamble

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **RFC 2119**.

Component Identity

We use Universally Unique Identifier (UUID) to identify a component.

Transportation Layer

The IoTtalk Data Gateway MUST broadcast its identity on IPv4 UDP 1900. We consider it act as a beacon. A beacon SHALL keep broadcasting in an constant interval. Taking one to five seconds as broadcasting interval is RECOMMENDED.

Protocol Grammar

The following ABNF grammar defines Component Discovery Protocol:

```
cd-protocol       = header body CRLF

header            = "IT" SP src-UUID SP

body              = 1*verb

verb              = hello
                  / ping
                  / pong

hello             = "HELO"
ping              = "PING" 0*1dest-UUID
pong              = "PONG" dest-UUID

src-UUID          = UUID
dest-UUID         = UUID
UUID              = 16OCTET
```

The sender MUST reveal its `UUID` in header.

The Protocol Commands

The *verb* described in section *Protocol Grammar* are considered as commands here.

`hello` Command

This command SHALL be used by a beacon to reveal its `UUID` at the local area network. It MUST carry a `UUID` refered to sender itself.

`ping` Command

This command is invoked when a component need to check the other known components still alive or not.

The following message format are valid:

- `PING UUID`
- `PING`

The former format carry a `UUID` refered to a destination component. If the `UUID` of the receiver do not match with message `UUID`, we MAY drop this message without response. Otherwise, the receiver SHOULD invoke a *pong Command*.

The later format do not make anything follow it. Any receiver SHOULD invoke a *pong Command* to original sender. In order to make components be discovered actively, this format is RECOMMENDED to send via broadcasting. Or in case of discovering all component on same endpoint, same ip, for example, this format is also RECOMMENDED.

`pong` Command

This command is invoked in order to reponse the *ping Command*.

The following message format is valid:

- `PONG UUID`

This message MUST carry a `UUID` referd to the source of *ping Command*. The receiver MAY drop this message if the carried `UUID` do not match with receiver itself.

Security Aspects

UDP Flood Attack

Due to this protocol use UDP as transportation layer. This Attack is a common issue. Attacker can send huge amount of udp packages with `ping` command to make every devices in the LAN busy.

We RECOMMENDED each device maintain a rate limitation for the `ping` command. Note again that the receiver of `ping` command SHOULD invoke response, but not MUST. Devices can drop udp packages if necessary.

Resource Access Protocol

In a LAN, our device and IoTtalk Data Gateway can known each other via *Component Discovery Protocol*. The device application which connects to IoTtalk always comes and go. We will consider them as kind of *resource*. They provide device feature (s) sometimes available and sometimes not.

In order to keep those resources well-managed, this specification will regulate the following *process*:

- Resource registration (creation)
- Resource deregistration (deletion)
- Resource metadata modification
- Resource metadata retrieval

  state Draft

Preamble

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **RFC 2119**.

Resource Identity

The basic unit of a resource is a device application. We will use UUID introduced in *Component Discovery Protocol* as resource identity.

The different process on same physical device MUST have different UUID. Because the feature on they may diverge.

Transportation Layer

The message delivered via the transportation layer MUST be constructed with two part:

- Header part
- Data part

The header part MUST indicate various protocol related metadata, including:

- The kind of *process* mentioned above.

- The URI endpoint for operating.

The data part is OPTIONAL and it indicate generic data container, including but can more than:

- The arguments for the *process*.
- Optional data field for the URI endpoint.

Implementation Suggestion

The RECOMMENDED transportation layer implementation is HTTP. The HTTP support is widespread, the implementation on devices will become easier.

In header part, the *process* can be mapped to the *HTTP Verb* like *GET*, *POST*, or *DELETE*. The HTTP URL is also easy to design. In data part, the JSON/XML content is suitable for deliver complex data structure.

Transportation Endpoint

The REST-like endpoint design on this protocol is RECOMMENDED.

Each endpoint has a valid set of *verbs*. The univeral set *verbs* is $\{\text{GET}, \text{POST}, \text{PUT}, \text{DELETE}\}$ which is inspired by HTTP verbs.

Registration Endpoint

This endpoint indicate the creation of a *resource*.

```
PUT /<id>

# body
<metadata>
```

Where `<id>` is the UUID of a resource mentioned in *Resource Identity*.

Where *<metadata>* section is OPTIONAL. The device can register first, then the *metadata* can be added later.

Rejection

We SHOULD reject the registration request if the *metadata* is not recognized.

Deregistration Endpoint

This endpoint indicate the deletion of a *resource*.

```
DELETE /<id>
```

Where `<id>` is the UUID of a resource mentioned in *Resource Identity*.

Metadata Retrieval Endpoint

This endpoint indicates the reading of resource *metadata*.

```
GET /<id>/<field locator>
```

Where `<id>` is the UUID of a resource mentioned in *Resource Identity*.

Where `<field locator>` is OPTIONAL. It indicates the selector of the data field, its format is implementation dependent.

Metadata Modification Endpoint

This endpoint indicates the updating of resource *metadata*.

```
POST /<id>/<field locator>

<metadata>
```

Where `<id>` is the UUID of a resource mentioned in *Resource Identity*.

Where `<field locator>` is OPTIONAL. It indicates the selector of the data field, its format is implementation dependent.

Rejection

We SHOULD reject the registration request if the *metadata* is not recognized.

Metadata

accept_protos List of supported protocols. This field is OPTIONAL.

idf_list *idf* stands for *Input Device Feature*. This field is OPTIONAL. It's a list of pair (*feature*, *units*).

- **Feature** It has naming convension: `([a-z][_a-z0-9]*)+`.
- **Units** It is a list of string or null value. We MUST infer the dimension of feature from shape of unit.

odf_list *odf* stands for *Output Device Feature*. This field is OPTIONAL. It's a list of pair (*feature*, *units*).

- **Feature** It has naming convension: `([a-z][_a-z0-9]*)+`.
- **Units** It is a list of string or null value. We MUST infer the dimension of feature from shape of unit.

name Arbitrary string, it can be consider as comment.

owner Arbitrary string. This field is OPTIONAL.

profile A JSON object for storing arbitrary data. This field is OPTIONAL.

Security Aspects

Resouce Control Protocol

The main purpose of *Resource Access Protocol* is for handling the metadata of resource. We also need some control signals for *changing the state of resource*. This signal was usually sent from the IoTtalk server. The state we concerned will be:

- Resource data channel connection state
- Resource data transfer state
- Resource unpluging

state WIP

Preamble

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **RFC 2119**.

Transportation Layer

The transportation layer MUST be bidirectional. The message can be fired by both server and client side. The passive server model, for example HTTP, is unqualified.

Implementation Suggestion

We RECOMMEND the Message Queuing Telemetry Transport (MQTT) or any kind of message queue protocol can achieve the bidirectional communication.

Control Signal

A single control signal on the transportation wire MUST be identity as a distinct, minimal unit. It has following structure:

- Signal type
- Signal payload: this field is OPTIONAL. It depends on the signal type.

The type reveals the main behavior of the signal will achieve. And the payload is for the required parameters about the signal.

The available types:

- Connect signal
- Disconnect signal
- Resume signal
- Suspend signal
- Shutdown signal

Connect Signal

This signal makes a resource setup its feature data channel with the proper parameters. The parameters of data channel MUST be revealed in the signal payload.

Disconnect Signal

This signal makes a resource reset its specific feature data channel connection and feature data transfer state. In other world, the resource will disconnect from data channel and disable the feature. The specific feature MUST be revealed in the signal payload.

Resume Signal

This signal is sent from IoTtalk that requests a resource feature start to do data exchange on the wire. The specific feature MUST be revealed in the signal payload.

Suspend Signal

This signal aims at stop the data transfer of specific resource feature on the wire. The specific feature MUST be revealed in the signal payload.

Shutdown Signal

This signal makes the resource fully shutdown. It will stop all data transfer, disconnect all from data channel, unregister itself, then maybe do some internal cleanup.

CHAPTER 4

API Specification

Resource Access API

Resource Access HTTP API

This specification will introduce HTTP API via REST-like design for *Resource Access Protocol*.

Endpoints

/<id>

GET /<id>

Retrieving the *metadata* of `<id>`.

Request Headers:

- **Accept**
  - *application/json; charset=utf-8*

Response Headers:

- **Content-Type**
  - *application/json; charset=utf-8*

Response JSON Object:

- id (*string*): The requested UUID.
- state (*string*): The state of the resource, *online* or *offline*.
- ... and other available metadata field.

Status Codes:

- **200 OK**
  - UUID exists
- **404 Not Found**
  - The requested UUID unknown

Request:

```
GET /219e0050-10e0-48dd-9b99-e196acfb30c8 HTTP/1.1
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "219e0050-10e0-48dd-9b99-e196acfb30c8",
    "state": "ok",
    "name": "BetaCat",
    "idf_list": [["meow", ["dB"]]],
    "accept_protos": ["MQTT", "WebSocket"],
    "rev": "cc35867e-1f74-47d3-88c9-7dcc374a5919",
    "profile": {
        "model": "AI"
    }
}
```

PUT /<id>

Request Headers:

- **Accept**
  - *application/json; charset=utf-8*
- **Content-Type**
  - *application/json; charset=utf-8*

Request JSON Object:

- name (*string*, optional): The name of the device application
- idf_list (*array*, optional): The Input Device Feature list of the device application
- odf_list (*array*, optional): The Output Device Feature list of the device application
- accept_protos (*array*): The accepted protocols list of the device application
- profile (*json*, optional): The data for device application details.

Response Headers:

- **Content-Type**
  - *application/json; charset=utf-8*

Response JSON Object:

- id (*string*): The requested UUID.

- state (*string*): The state of the resource, *ok* or *error*.
- reason (*string*, optional): The error message.
- rev (*string*): the token required by deregistration. It stands for *revision*.
- url (*json*)
- ctrl_chans (*array*):We use two mqtt topics here, in order to achieve bidirectional communication. The `i` topic denote the uplink, client can send control channel request via this link. Also, the `o` topic denote the downlink, server will send control command via this channel.

Status Codes:

- **200 OK**
  - UUID registration accepted.
- **400 Bad Request**
  - Wrong *Content-Type*.
- **403 Forbidden**
  - Any content of metadata is not recognized.

Request:

```
PUT /219e0050-10e0-48dd-9b99-e196acfb30c8 HTTP/1.1
Accept: application/json

{
    "name": "BetaCat",
    "idf_list": [["meow", ["dB"]]],
    "accept_protos": ["MQTT", "WebSocket"],
    "profile": {
        "model": "AI"
    }
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "219e0050-10e0-48dd-9b99-e196acfb30c8",
    "rev": "41997b1e-2850-43b5-b4b5-309d05307bf7",
    "state": "ok",
    "url": {
        "scheme": "mqtt",
        "host": "example.org",
        "port": 1883
    },
    "ctrl_chans": [
        "219e0050-10e0-48dd-9b99-e196acfb30c8/ctrl/i",
        "219e0050-10e0-48dd-9b99-e196acfb30c8/ctrl/o"
    ]
}
```

Error Response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "id": "219e0050-10e0-48dd-9b99-e196acfb30c8",
    "state": "error",
    "reason": "feature not supported"
}
```

DELETE /<id>

Request Headers:

- **Accept**
  - *application/json; charset=utf-8*
- **Content-Type**
  - *application/json; charset=utf-8*

Request JSON Object:

- rev (*string*): the token required by deregistration. It stands for revision.

Response Headers:

- **Content-Type**
  - *application/json; charset=utf-8*

Response JSON Object:

- id (*string*): The requested UUID.
- state (*string*): The state of the resource, *ok* or *error*.
- reason (*string*, optional): The error message.

Status Codes:

- **200 OK**
  - UUID successfully unregistered.
- **400 Bad Request**
  - Wrong *Content-Type* or *revision* out-of-date.
- **404 Not Found**
  - UUID already unregistered or not found.

Request:

```
DELETE /219e0050-10e0-48dd-9b99-e196acfb30c8 HTTP/1.1
Accept: application/json
Content-Type: application/json

{
    "rev": "41997b1e-2850-43b5-b4b5-309d05307bf7"
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "219e0050-10e0-48dd-9b99-e196acfb30c8",
    "state": "ok"
}
```

Error Response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "id": "219e0050-10e0-48dd-9b99-e196acfb30c8",
    "state": "error",
    "reason": "id not found"
}
```

Resource Control API

Resource Control MQTT API

This specification will introduce MQTT API for *Resouce Control Protocol*

A control message has following basic skeleton:

```
{
    'msg_id': '...',
    'idf|odf': 'feature_name',
    'command': 'CMD',
    'topic': '...',
}
```

And the expected response:

```
{
    'msg_id': '...',
    'state': 'ok'
}
```

Most of the control message is *device feature* level.

Signals

Connect Signal

Example:

```
{
    'msg_id': '99dd9b65-f7cf-4219-a4b2-60bc16e79670',
    'idf': 'meow',
    'command': 'CONNECT',
    'topic': 'iottalk/esm/5289c32a-bcb3-434d-80b6-de3a22dfc746/i'
}
```

Response:

```
{
    'msg_id': '99dd9b65-f7cf-4219-a4b2-60bc16e79670',
    'state': 'ok'
}
```

Disconnect Signal

Example:

```
{
    'msg_id': 'fd8d118c-11a0-4dc5-b8fe-e53a41149b09',
    'idf': 'meow',
    'command': 'DISCONNECT',
    'topic': 'iottalk/esm/5289c32a-bcb3-434d-80b6-de3a22dfc746/i'
}
```

Response:

```
{
    'msg_id': 'fd8d118c-11a0-4dc5-b8fe-e53a41149b09',
    'state': 'ok'
}
```

Resume Signal

Example:

```
{
    'msg_id': '9ea799c9-707b-4107-9903-686aa393e96d',
    'idf': 'meow',
    'command': 'RESUME',
}
```

Response:

```
{
    'msg_id': '9ea799c9-707b-4107-9903-686aa393e96d',
    'state': 'ok'
}
```

Suspend Signal

Example:

```
{
    'msg_id': '3760aecf-e009-4c14-9a74-7c1800611763',
    'idf': 'meow',
    'command': 'SUSPEND',
}
```

Response:

```
{
    'msg_id': '3760aecf-e009-4c14-9a74-7c1800611763',
    'state': 'ok'
}
```

CHAPTER 5

Design of Grahpical User Interface

Concept

Model

Model denote a set of features. It has naming convension: `([A-Z][a-z0-9]*)+`.

The basic model is the whole set of features. If we add a device with the basic model, we will be asked to select which features are desired.

If we want to make an alias of a set of features, we can create a custom model.

Index