IoT Platform User Guide
(Version 5_0.0)

<Revision History>

Version No.	Description	Date
Version 1.0	First version	2016/10/07
Version 1.1	Updates based on K5 Launch on UK Site	2016/11/01
Version 1.2	Error correction Maximum number of Resources that can be related to a single Access Code has been specified	2016/12/01
Version 1.2	Error correction	2017/01/12
Version 1.3	Error correciton	2017/04/12
4_1.0	-	Internal Version (Not released)
4_2.0	Added notes on feature extensions 2.1.1. Added notes on data collection and functions related to P permissions and G permissions 5.1. Added section on information provided with registered data 5.3. Added section on the batch acquisition of data spanning multiple resources 5.5. Added section on the frequency resources are created using "CDL" permissions 5.8. Added section on client authentication (feature to be provided from July 2017) 5.9. Added section on using CORS 5.10. Added section on referencing logs when events and transfers fail 8.8. Added note on the ability to use G permissions and P permissions in the example of referencing multiple resource data entries with a single API operation ($all) 8.9. Corrected "key" to "name" in the example concerning only referencing a specific key in resource data (select) 10.3. Added note on Range Request in the Memo column in the example concerning referencing Resource_Binary data 13.1. Added Content-Type to the legend in the example concerning registering events 13.2. Corrected errors in relation to the header field value in the example concerning referencing events	2017/08/01
4_2.1	5.8. Added description about Example of Client certificate	2017/10/11
5_0.0	Added notes on feature extensions 5.11. Added section on JSON conversion 8.10. Added example of registering csv/txt/binary/gzip file in "Resource_JSON" and referencing the JSON data 8.11. Added example of retrieving with specifying elements position of the array	2017/11/15

Preface

Thank you for considering the "IoT Platform".
This "IoT Platform User Guide (hereafter, this manual)" is intended for customers considering or implementing this service. We ask for your understanding regarding the following matters.

1. Customers considering this service are kindly requested to utilize this manual to assist in making their decision to implement this service only.
2. This manual and the contents therein are not to be disclosed or provided to any third parties.
3. Copying or reproducing the contents of this manual without the permission of the provider is prohibited.

This manual contains important information to be used in implementing this service.
Customers signing up to a service contract are asked to thoroughly read this manual prior to using this service.
Please handle this manual with care and store it in a safe place.
Customers opting to not use this service are responsible for promptly disposing of this manual.

While we have striven to prepare this manual with the utmost of care in describing tasks in the most easy-to-understand manner as possible, we cannot be held responsible for any errors or omissions in the content of this manual. This manual and the contents therein may change at any time without notice.

The contents of this manual cannot be copied, reproduced or modified, in part or in full, without prior permission to do so.

Disclaimers

- We do not accept any responsibility for unexpected malfunctions or for unforeseen charges occurring due to the user performing operations not listed in this manual of this service.
- We do not accept any responsibility in the unlikely event that the use, or inability to use, this service causes damage to the user (including, but not limited to, damages caused by a suspension of work, damage to/loss of data, or accident-related damages, and including the potential for liability claims from a third party).

<Terms>

Term	Description	Notes
IoT	An abbreviation of "Internet of Things" This refers to "things" connected to a network via an Internet protocol (Internet language)".	
REST	An abbreviation of REpresentational State Transfer This refers to a software design format where design principles optimized for linking multiple software instances are adapted for the web. *A caller interface (referred to as a "RESTful API") sends messages written in XML to a specific URL via HTTP(s). When using this service replies are sent back in JSON format, not XML.	
MQTT	An abbreviation of Message Queuing Telemetry Transport MQTT is a light communications protocol on the TCP/IP network suited to frequently sending and receiving short messages between multiple subjects. The use of MQTT is now very common in M2M networks and the IoT (Internet of Things) field. At a minimum the header is only two bytes in size, significantly reducing the amount of communications traffic, CPU load and energy consumption required, compared to sending the same communications via HTTP.	
Dynamic resource controller (DRC)	Dynamic Resource Controller provides distributed control processing based on proprietary wide-area distribution technologies. This function assists in the optimal collection of data from a limited number of resources based on traffic fluctuation during data collection.	
Resources	The collection unit for IoT data	
Resource data	One piece of data	
Access codes	Authorization information for resources	

- Contents -

Chapter 1 Introduction

1.1 Purpose of this Manual

This manual is an user guide intended for persons developing applications using APIs based on the use of the IoT Platform (hereafter, this service).

1.2 Available Documents

The following manuals have been prepared to support customers using this service.

Manual name	Description
IoT Platform Service Details Instruction Manual	Describes service specifications.
IoT Platform User Guide	A manual used to support API usage during application design and with this service, including specific examples. (This manual)
IoT Platform API Reference	An API reference manual used for application design using this service.
IoT Platform Service Portal Operating Manual	A manual describing web interface functionality (hereafter, the Service Portal).

Memo

Refer to Chapter 3 of the "IoT Platform Service Portal Operating Manual" for more information about resources, access codes and other general service definitions and concepts

1.3 Features

The APIs used by this service, feature the following characteristics designed to support your IoT business initiatives.

- Easy-to-use interface

  This service is compatible with the widely used REST (HTTP) and the lightweight, low-intensity MQTT protocols. An interface for these protocols can be easily applied for your business, and is readily available.

- Deep searching tools includea vast array of query options

  Many OData-like queries are available for using resource data. Combine these queries to perform searches using complex conditions.

- Compatible with SSL/TLS for secure data access

  Data can be accessed in a secured way with SSL/TLS-compatible APIs. Access permissions can be set for each data group (resource) to allow for secure data use between multiple affiliate companies.

Chapter 2 Feature Highlights

2.1 Feature Highlights of IoT-PF

2.1.1 Data Collection

"Resource", a data registration unit, is created within the customer tenant. “Access codes” granting access permissions to the resource must be created and assigned to the resource.

Assign access codes to the API to approve and run the requests the access code corresponds to.

<Permission types>

C: Permission to create resources, access codes, etc.

R: Permission to register data

U: Permission to reference data

D: Permission to delete resources, access codes, events, etc.

L: Permission to acquire a list of resources, access codes, events, etc.

P: Permission to register data to a specific directory under the resource

G: Permission to reference data from a specific directory under the resource

2.1.2 Data Collection Preparation

Operations can be performed from the Service Portal, or with an API.

*API operations can only be performed for “resources”, “access codes”, and “events”.

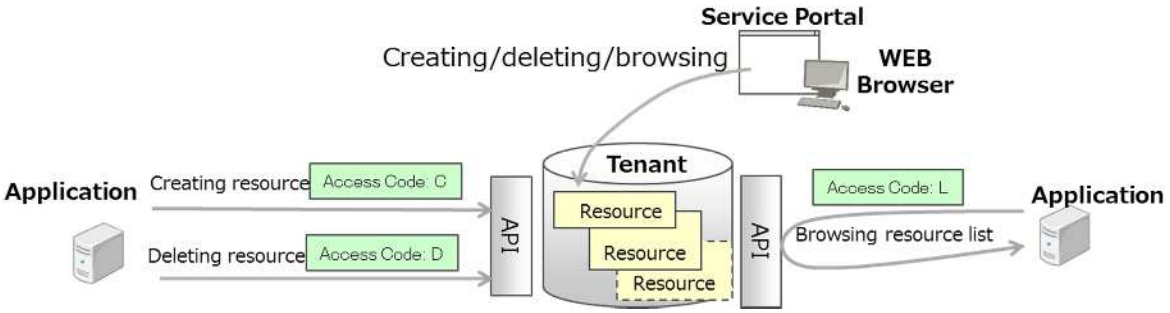


Figure 1 Managing resources

2.1.3 Collecting and using data

Data registration and browsing can be executed with API.

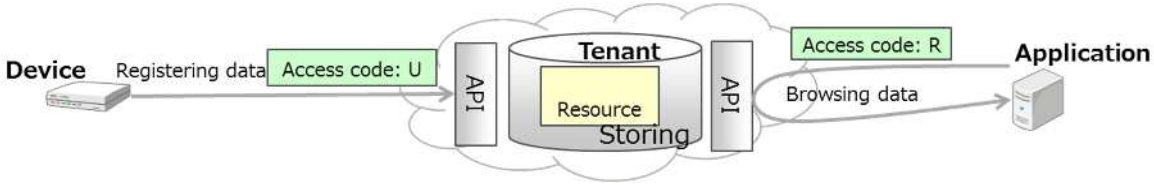


Figure 2 Storing data

Data can be transferred to other services without the need to store the data with this service.

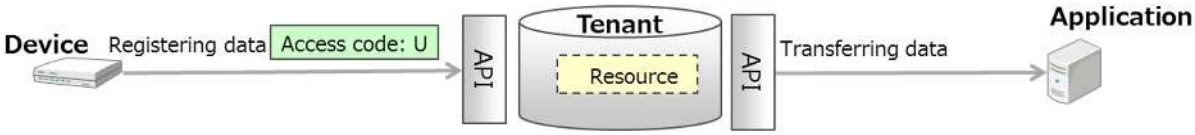


Figure 3 Transferring data

2.1.4 Event functionality

Set conditions to extract data as events and perform actions when conditions are met based on data registration/update triggers. These actions can be configured to send email notifications, or start up a specific API.

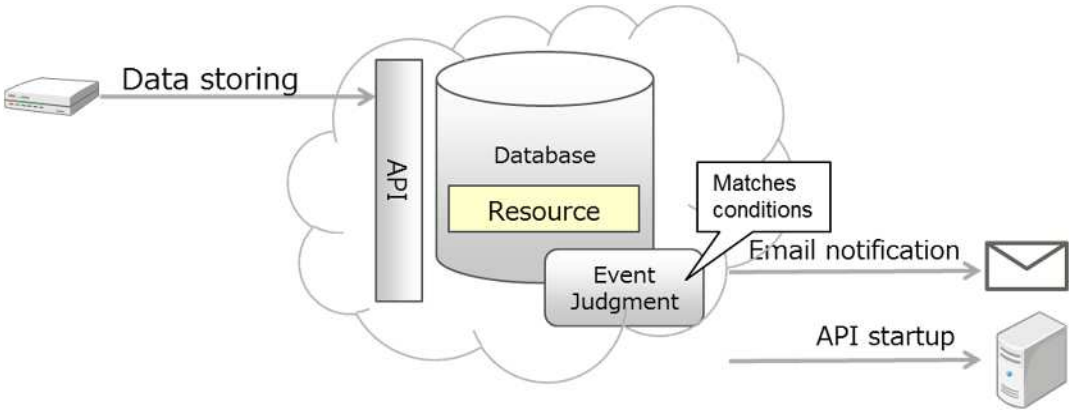


Figure 4 Event functionality

2.1.5 Access Restrictions

Restrictions can be set for IP addresses accessing this service.
You can restrict access to resources to both the Service Portal and access codes.

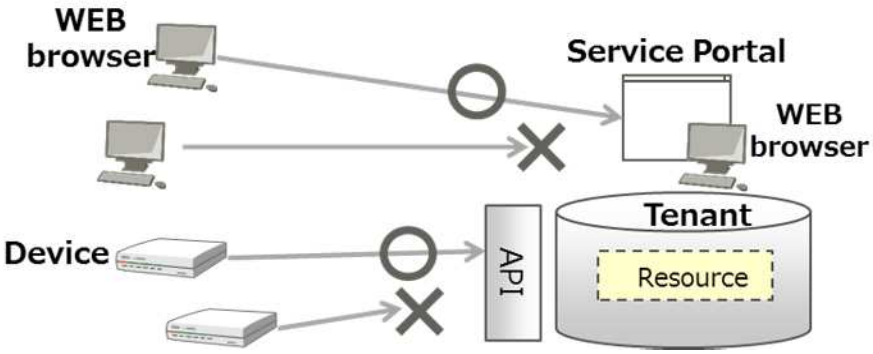


Figure 5 Access restrictions

2.2 Dynamic Resource Controller

This provides recommend resources, available for use during data collection, as a means of controlling location of data collection based on cloud load. This allows data collection to be optimized to the level of IT resources available to the customer.

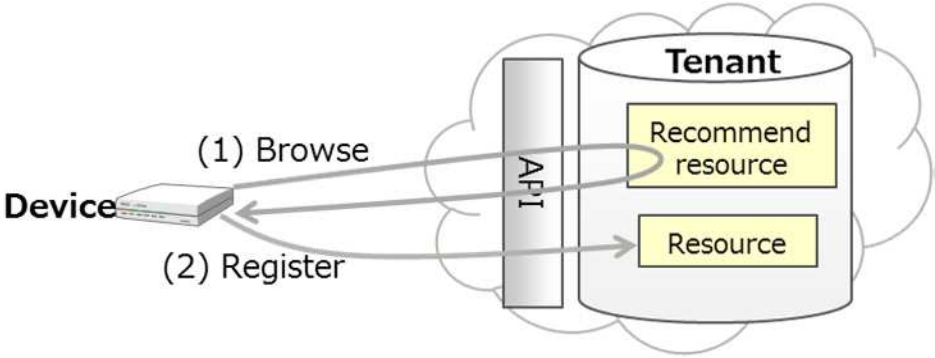


Figure 6 Dynamic resource controller

2.2.1 Basic Operation

The basic operation of the dynamic resource controller is as follows.

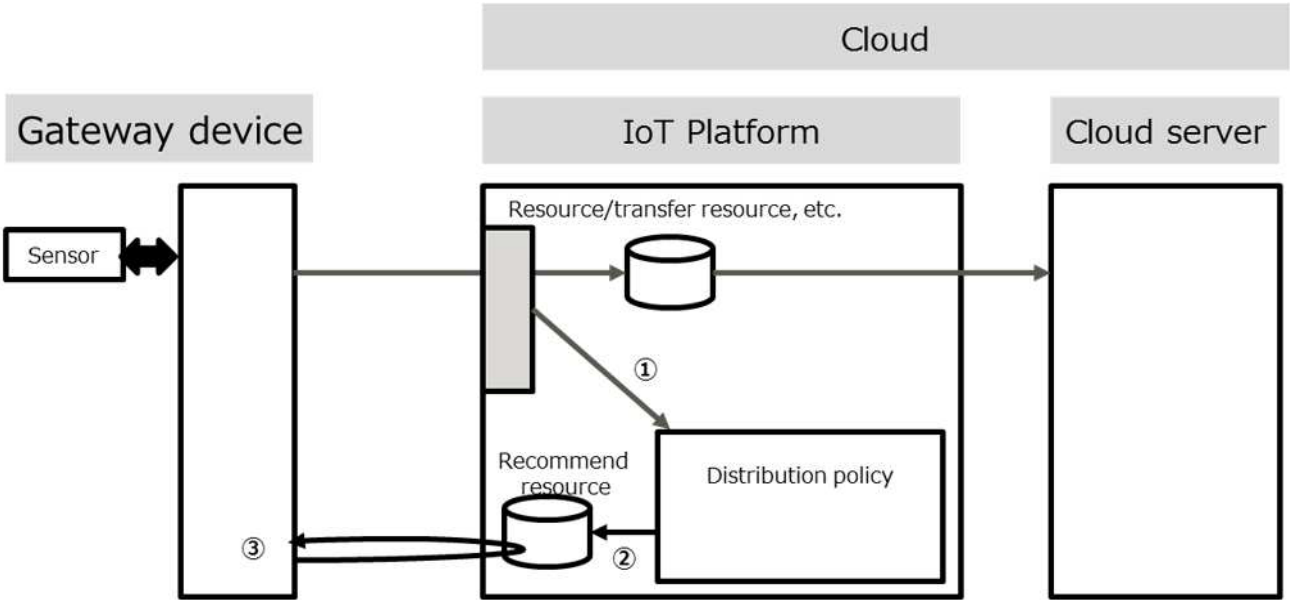


Figure 7 Basic operation of the DRC

This service collects service load information and compares these values with conditions set by the customer (edge computing conditions) to provide recommendations to the customer. Customers can develop applications to optimally collect data within the IT resources available using these recommendations.
The process flow for using the DRC is as follows.

(1) This service monitors customer tenant load.
(2) This service flags recommend resources based on a pre-defined distribution policy.
(3) Applications on the gateway device modify processes based on the rewritten recommend resources.

This enables processes to be modified based on system load, allowing for stable system operation.

2.2.2 Distribution Policy

A distribution policy is a group of settings which determine the manner in which the DRC works.
These include settings for the recommend cycles, calculation period and resources for measuring loads.
Additional settings include the target recommend resources, edge computing conditions and the removal conditions for publishing recommends.
These settings are used to compare load information with edge computing conditions set by the customer, and write the results of conditional matches to the recommend resources.

2.2.3 Load Measurement

Load measurements write the average load to the load resource, using intervals set for the “recommend cycle”, to calculate the average for the “calculation period”. Sudden variations in load can be smoothed out by making the calculation period longer than the recommend cycle.

2.2.4 Recommend Resource

An application, deployed to the gateway device, references recommend resources.
If the recommend value (recommend_value) is ON, the gateway device will run processes that are normally performed on the cloud server, reducing the communication traffic and cloud server load.

This section describes recommend JSON data used for the recommend resource.

Table 1: Description of JSON key names and values for the recommend resource

Key name	Description of value
recommend_value	The recommend value (“ON” or “OFF”)
recommend_parameter	The recommend parameter (character string) (The parameter set by the distribution policy. However, “ALM” is used when there is an extreme system load placed on this service)

The following example shows JSON data stored to the recommend resource when the recommend value is changed to “ON”, and the edge computing condition parameter is “recommend is on”.

```
{
   "recommend_value" : "ON”
"recommend_parameter" : "recommend is on"
}
```

2.2.5 Load Resources

You can view information on the load by referencing the load resource. This section describes the load JSON data stored to the load resource.

Table 2: Description of JSON key names and values for the load resource

Key name	Description
tps	The load data on the customer tenant (tps) [transaction/second]
bps	The load data on the customer tenant (bps) [bit/second]

The following example shows JSON data stored to the load resource if the load data is 10 tps and 10,000 bps.

```
{
    "tps" : "10",
   "bps" : "10000"
}
```

Develop an application so that the above JSON data can be interpreted.

Chapter 3 Preparations

The following items are required for app development.

1. Technical information provided after service contract

Following information which will be provided after service contract will be necessary. The MQTT password can be configured from the Service Portal.

Table 3: List of technical information provided after contract

	Item
1	Tenant ID
2	MQTT broker address
3	Base URI
4	MQTT user name

2. App development environment

Either a REST (HTTP) library or an MQTT client library is required to develop data collection and referencing apps. For a more detailed description of the operating environment requirements that apply, check the specifications for the OS and programming language used for development.

The descriptions provided in this manual are based on the curl command in the Linux OS and Python language scripts that incorporate a MQTT client module.
The method used to install the curl command in Ubuntu is described below. (Performed with su permissions)

```
# apt-get install curl
```

The method used to install Python and the Eclipse Paho module (the MQTT client) in Ubuntu is described below. (Performed with su permissions)

```
# apt-get install python
# apt-get install python-pip
# pip install paho-mqtt
```

3. Root certificates (when using SSL/TLS)

This service uses "DigiCert SHA2 High Assurance Server CA" provided by DigiCert, Inc. for SSL/TLS server certificates.

Please use "DigiCert High Assurance EV Root CA" from DigiCert Trusted Root Authority as the root certificate when using SSL/TLS communications.

"DigiCert High Assurance EV Root CA" can be obtained from the following address.

https://www.digicert.com/digicert-root-certificates.htm#roots

*This URL link was tested as active as of September 2016.

Chapter 4 Flow of System Configuration

The following is the flow of system configuration using this service.

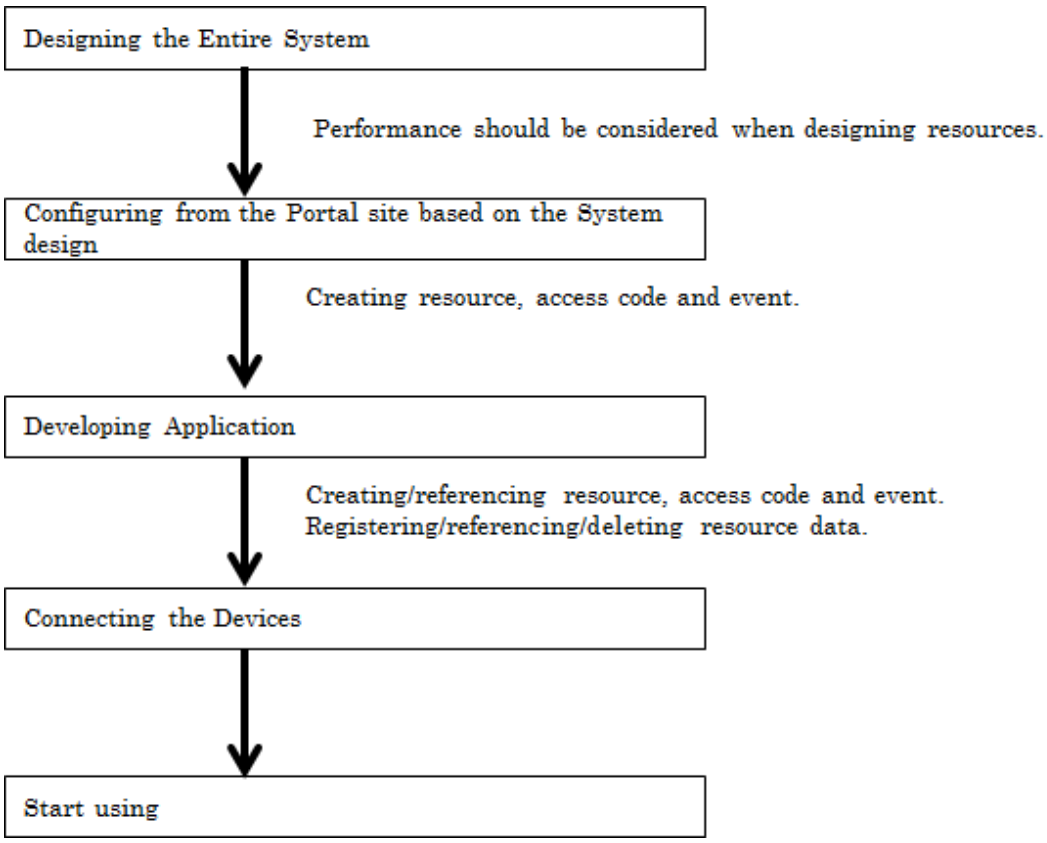


Figure 8 Workflow of the system configuration

When configuring a system, designing the entire system before developing the application becomes important in order to secure the performance and security. Please refer to Chapter 5 for the concept of design.

Chapter 5 Concept of Design

This chapter covers matters that require additional care and caution when committing to designs. Read through this chapter carefully when designing resources, access codes, and events.

5.1 Information Provided with Registered Data

This service adds essential parameters to registered JSON data when storing to distinguish data for referencing. The information added to data will vary depending on the function used. The chapter describes the information provided with references to specific examples.

Registration Using JSON

The resource path and registration date will be registered using REST.
Refer to the example provided here. First, create the following resource and access code.

Resource type	Resources
Resource path	area1/device1
Data format	JSON
Retention period	One day

Access code	DeviceAccessCode
Resource path	area1/device1
Access permission	Add permissions for U and R

Next, publish the following API and register the JSON syntax.

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer DeviceAccessCode
URI	http://<zone>.fujitsu.com/v1/<Tenant ID>/area1/device1.json
BODY	{ “sensor1": 100}

Check that 200 OK is returned, and then call the following API and reference the JSON syntax.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer DeviceAccessCode
URI	http://<zone>.fujitsu.com/v1/<Tenant ID>/area1/device1/_past

The JSON syntax written in the BODY part of the response is as follows. The original registered JSON data is the name “_data” value, and the resource path (_resource_path) and registration date (_date) are added as names.

```
[
 {
  "_data":{
     "sensor1": 100
  },
   "_date": "20170214T072509.987Z",
   "_resource_path": "area1/device1"
 }
```

```
]
```

Refer to Section 3.3 of the API Reference Guide for more details concerning the response body syntax.

Registration Using Events (Mail)

When sending an email using an event, an attachment file (eventinfo.txt) will be added to the email containing JSON data registered to email attachments, and details of operations performed, the date, event ID, and resource path.

The following text file will be attached to emails if an event has been set to skip emails for the resource described in the preceding section below. The contents of "eventinfo.txt" are as follows. The original JSON data registered is the "body" value, and operation details (operation), the event publication date (date), event ID (event_id) and the resource path (resource_path) are added.

```
{
   "operation":"create",
   "date":"20170214T072509.99Z",
   "body": {
                "sensor1" : 100
              },
   "event_id":"58a2b0bc0341",
   "resource_path":"area1/device1"
}
```

Refer to Section 10.2 of the API Reference Guide for more details concerning the settings of Email notification.

Registration Using Resource (Transfer)

Only the original JSON data is sent to the API recipient when calling an external API using Resource (Transfer). Refer to the example provided here. First, create the following resource and access code. The external API is to be prepared by the customer.

Resource type	Resource (transfer)
Resource path	_fwd/area1/device1
Data format	JSON
URI	http://<External API>
Method	PUT

Next, assign the following access code and access permissions for the resource path below.

Access code	FwdDeviceAccessCode
Resource path	_fwd/area1/device1
Access permission	Add permissions for U and R

Next, publish the following API and register the JSON syntax.

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer DeviceAccessCode
URI	http://<zone>.fujitsu.com/v1/<Tenant ID>/_fwd/area1/device1.json
BODY	{ "sensor1": 100}

Check that 200 OK is returned, and then reference the JSON syntax received from the external API server at the

transfer destination. The JSON syntax written in the BODY part of the response received by the server with the external API is as follows. This is the same as the request JSON syntax.

```
{ “sensor1": 100}
```

5.2 Resource_JSON Performance

Errors in the “Resource_JSON” design may result in longer response times and may make it more difficult to extract data. The resource design method described below is aimed at improving usability. .

This explanation will use the following example. Operation may become more difficult when consolidating data from multiple terminals onto a single resource, as shown in the figure below, due to the inability to acquire data once the maximum data volume is exceeded, or due to data retrieval taking an enormously long time, etc.

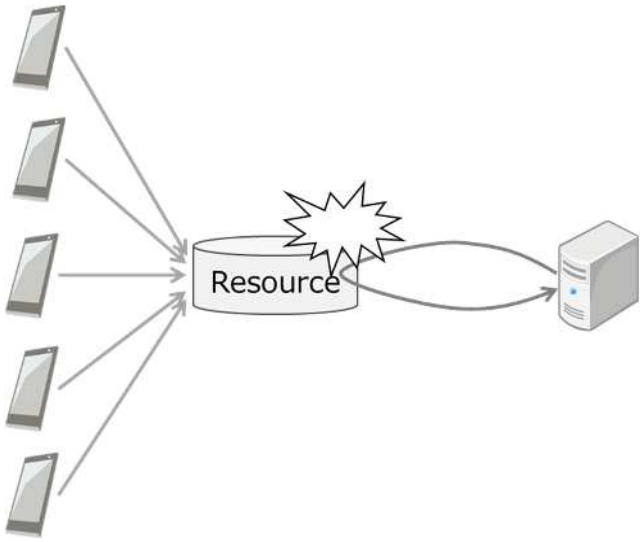


Figure 9 Non-recommended example

The above-mentioned issues can be avoided by creating a single resource for each terminal and sensor, as shown in the figure below. System requirements must be considered when implementing system designs. With “Resource_JSON”, you can also browse multiple resource data entries with a single API operation. Refer to Chapter 8.8 (of this guide) or Chapter 3.3 of the “IoT Platform API Reference” for more details.

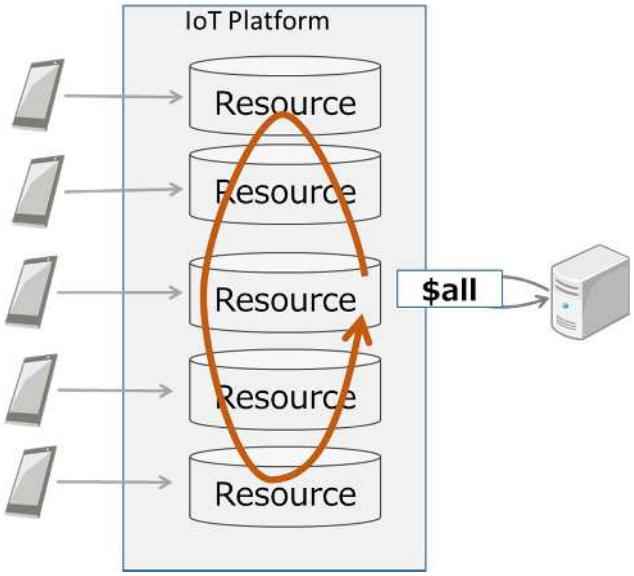


Figure 10 Recommended example (referencing multiple resources by a single API operation)

5.3 Batch Acquisition of Data Spanning Multiple Resources

Using access code "R" permissions to acquire data from multiple resources simultaneously, as described in Chapter5.2, requires that the same access code be granted to all corresponding resources. Note that any data acquisition attempt will fail if even one of these resources lacks the access code required.

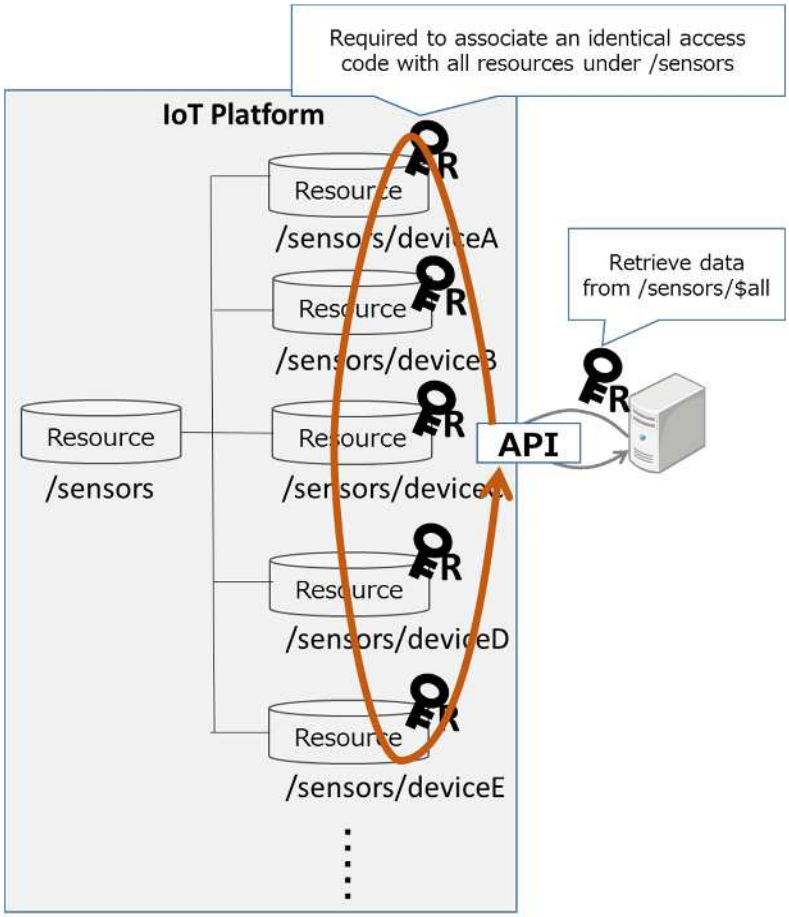


Figure 11 Using access code "R" permissions to reference multiple resources

This problem can be resolved using access code "G" permissions. Set access code "G" permissions to a resource as shown in the figure below to reference all resource data under a set resource. Refer to Chapter 8.8for more details.

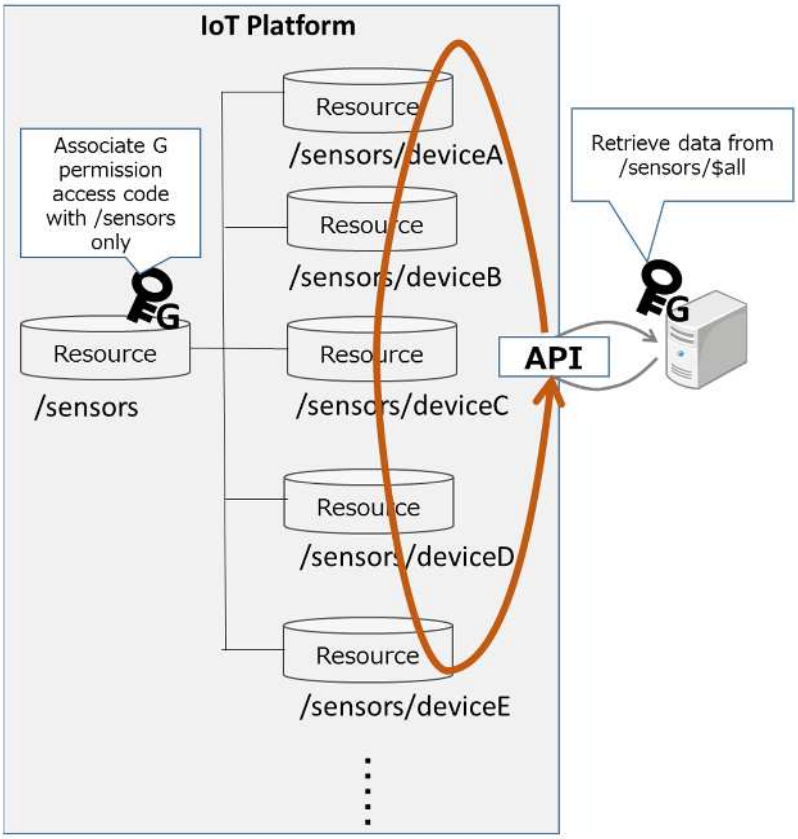


Figure 12 Using access code "G" permissions to reference multiple resources

5.4 Setting " CDL" permissions in access code

When " CDL" permission is set to multiple Resources and metadata reference of resource is executed, the response might become slow as in such case retrieving takes place under all the specified resources.
So, please do not set the "CDL" permission to a lot of Resources when configuring a single access code.

5.5 About Frequency of creating Resources Using "CDL" Permissions

While resources can be created and deleted using access codes with "CDL" permissions, repeated use of this to create and delete numerous resources over the course of a day should be avoided. Doing this may cause system performance to degrade on a whole.

5.6 Setting Server when using Transfer or Event feature

When designing an API different from this service by using Transfer or Event function, please turn on KeepAlive on API receive server.
In case of KeepAlive is off (not turned on), Transfer or Event process might end before the data processing is completed, and result data loss.

5.7 Scope of Disclosure for Resources

Take the scope of disclosure into consideration when designing resources and access codes. For example, having sensor data for various users stored on a single resource is undesirable from a security standpoint, as users will be able to view other user's sensor data.

5.8. Client Authentication

Client authentication enables authentication on a device basis, allowing for securer communications. Perform client authentication as follows.

1) Generate a client certificate

Generate a file that contains a client private key and client certificate as a set.
Refer to Chapter 4.13 of the "Service Portal Operating Manual".

2) Separate the client certificate

Separate the client private key and client certificate from the file.

3) Configure access code settings

Set the client certificate prepared beforehand to the access code.
Refer to Chapter 4.7 of the "Service Portal Operating Manual" for more information on configuration settings.

4) Set the client certificate to the device

Store the client private key and client certificate to the device used to register and reference data.

5) Publish API

Publish the API. Reference the private key and client certificate when publishing.

Next, we describe about the procedure of separating the Client private key and the Client certificate from PKCS#12 from file (client.p12) acquired from the service portal, together with the example.
The CA certificate, the Client certificate, and the Client private key of this service are stored in the PKCS#12 file as a set.

In order to set the client authentication to the access code, the Client certificate is needed. Moreover, to issue API by using that access code, the Client certificate and the Client private key are needed. So, it is necessary to separate them from the original PKCS#12 format file. (However, depending on the REST application, it might be possible to execute them without separating from KKCS#12 format file.)

Here in this example, we describe the method of generating client.crt (Client certificate) and client.key (Client private key) after separating from the PKCS#12 format file.

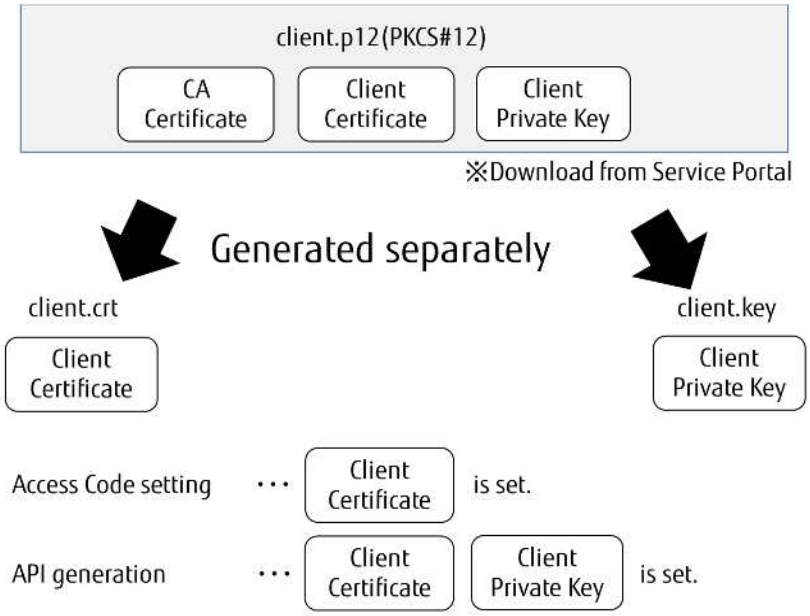


Figure 13 Separating PKCS#12

OpenSSL command needs to be installed as a pre requirement. Please refer to the official OpenSSL site about the method of installation. In this chapter it is explained by using the example of execution on Ubuntu.
First of all, Client certificate (client.crt) is generated from the PKCS #12 file (here 20171011000000.p12 is used as an example) by executing the following command.

```
openssl pkcs12 -in 20171011000000.p12 -clcerts -nokeys -out client.crt
```

Please set client.crt that is generated here to the access code. Please refer to Chapter 4.7 of the Service Portal Operating Manual for the setting method of the access code.

Next, Client private key (client.key) is generated from PKCS #12 file (20171011000000.p12) by executing the following command.

```
openssl pkcs12 -in 20171011000000.p12 -nocerts -nodes -out client.key
```

Please use client.key and client.crt when issuing API. It is possible to execute it by using the argument "-- key" and "-E" when executing it with curl. In the following example, the Client certificate is set to access code ClientAccessCode, and API is issued by using this access code.

```
curl -X PUT -H 'Authorization : Bearer ClientAccessCode' 'https://<zone>.fujitsu.com/v1/<Tenant ID>/clientResource.json' -d '{"sensor": 10}'  -E <file path for client.crt>/client.crt  --key <file path for client.key>/client.key
```

5.9. Using CORS

Normally you will not be able to access APIs when using JavaScript and other functions.
Enable CORS (Cross-Origin Resource Sharing) settings to allow for communication using JavaScript. Refer to Section 4.14.2 of the "Service Portal Operating Manual" for more information on configuration settings.

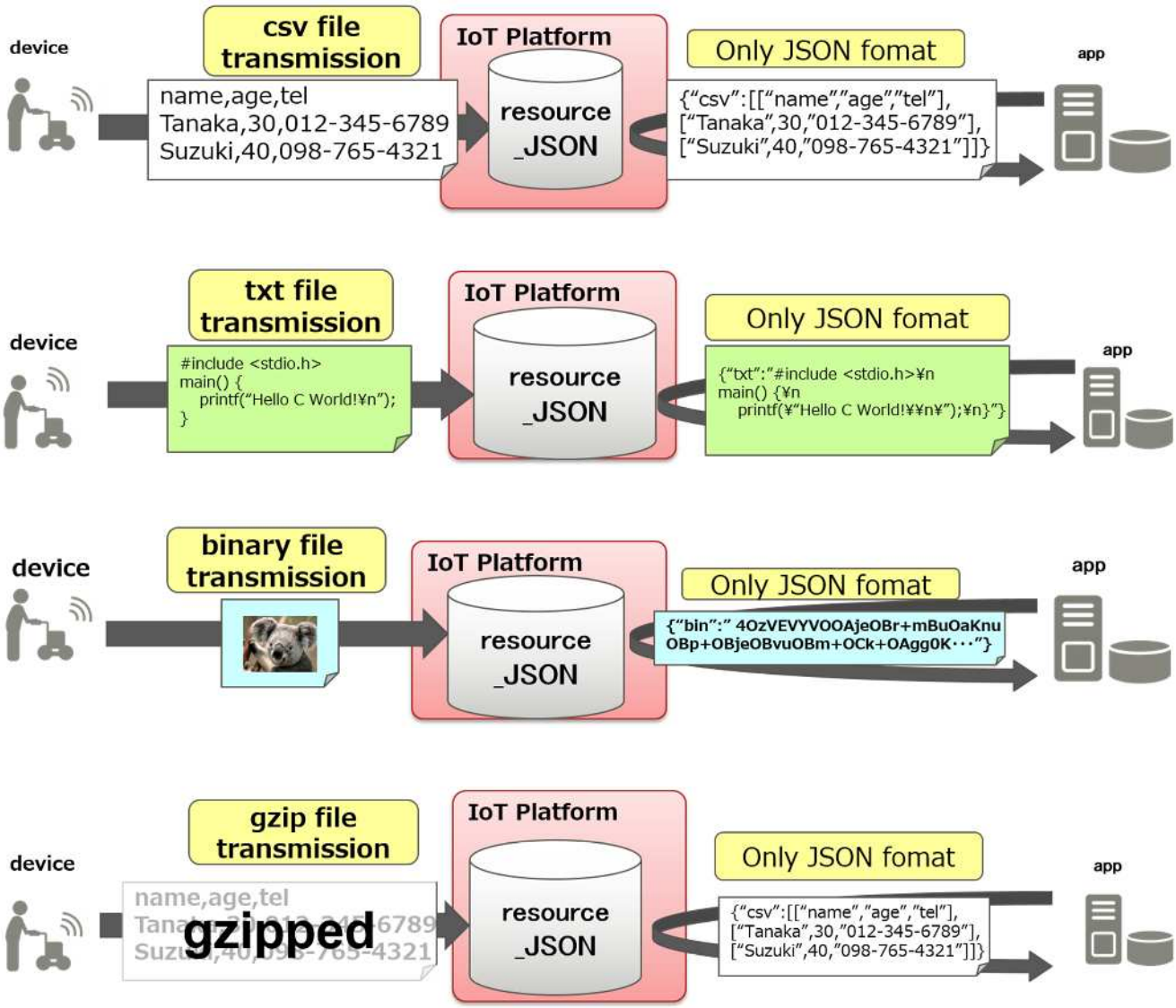
5.10. Referencing Logs When Events and Transfers Fail

Events, resource (transfer) and MQTT can be used to connect to a separate server. Use this function when communications with your server are down due to some unknown reason to confirm the causes of the error by referencing the error collection resource "_error".
An access code must be set for the error collection resource "_error". Refer to the "Service Portal Operating Manual" for more information on configuration settings. Refer to the API Reference Guide for more details on data stored to the error collection resource.

5.11. JSON conversion

In this service, a csv/txt/binary/gzip file can be registered in "Resource_JSON". The registered data is stored in the DB after JSON conversion by this service. Compressed gzip format csv file etc. is converted to JSON format after decompression and stored in this service.

Please be noted that the registered data can be reffered/searched only in JSON format. In case of using this function, please pay attention to this when designing the system.



Chapter 6 Concrete Example of System Design

The ethod of system designing is described below using an example of collecting data from multiple sensors in the Kamata area of Tokyo. This example assumes that number of sensors may change dynamically according to the circumstances and the user prefers to add or update resources by API when the situation calls for it.

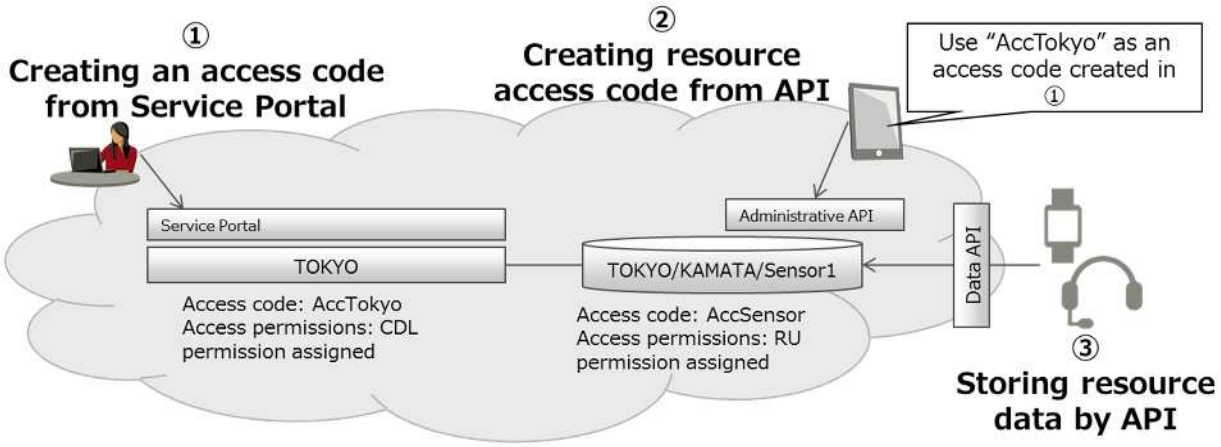


Figure 6 Example of System Design

(1) Create resources, and their access codes from the Service Portal

First, create the necessary resources, and the access codes with CDL permissions for such, from the Service Portal. In this example we will create the following.

Resource path of the parent resource created	TOKYO

Access code created	AccTOKYO
Resource specified as Resource 1	TOKYO
Access permissions for the resource specified as Resource 1	CDL permission assigned

(2) Create resources, and access codes with an API

Next, we use an API to create the following resource and access code for storing sensor information in Kamata.

Resource created	TOKYO/KAMATA/Sensor1

Access code created	AccKamataSensor
Resource specified as Resource 1	TOKYO/KAMATA/Sensor1
Access permissions for the resource specified as Resource 1	UR permission assigned

The API parameters used when creating access codes are described below.

Refer to Chapter 11.1 of this manual and Chapter 7 of the "IoT Platform API Reference" for more details on the contents of the BODY text.

Method	POST
Header field name	Authorization : Bearer AccTOKYO
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/ TOKYO/KAMATA/Sensor1

A 201 Created message will be returned when the resource has been created successfully.
In this case, the access code with CDL permissions ("AccTOKYO") must be specified for the parent resource.
Similarly, the API parameters used when creating access codes are described below.
Refer to Chapter 12.1 (of this guide) and Chapter 8 of the "IoT Platform API Reference" for more details on the contents of the BODY text.

Method	POST
Header	Authorization : Bearer AccTOKYO
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_access_code/AccKamataSensor

A 201 Created message will be returned when the access code has been created successfully.
In this case, the access code with CDL permissions ("AccTOKYO") must be specified for the parent resource.

(3) Store resource data to the resource

Now, resource data can be registered to "TOKYO/KAMATA/Sensor1" in the following manner. Refer to Chapter 8.1 (of this guide) or Chapter 3.1 of the "IoT Platform API Reference" for more details on the contents of the BODY text.

Method	PUT
Header	Authorization : Bearer AccKamataSensor
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/TOKYO/KAMATA/Sensor1

A 200 OK message will be returned when the resource data has been registered successfully.

Chapter 7 Specificational Restrictions (Design Precaution)

7.1 APIs

7.1.1 Compatible Protocols

This service supports the REST (HTTP) and MQTT protocols as an interface for registering, updating, referencing and deleting (hereafter, using) data.

As the REST (HTTP) protocol is widely used, it can easily be applied to a system which is already running it.

MQTT is a more lightweight protocol when compared to REST. Using MQTT when frequently sending sensor data can help alleviate power consumption and CPU load for the device (smartphone, gateway device, etc.). Eclipse Paho and other clients are available.

Further, MQTT has a PUBLISH/SUBSCRIBE function. SUBSCRIBE can be used to create an app that sends notifications to, and controls the device.

SSL/TLS communications can be used when using APIs with this service. Communicating with SSL/TLS allows for secure and safe data access.[1]

7.1.2 Non-guarantee of Transactions

This service utilizes an asynchronous architecture to deliver better performance, and does not guarantee transactions when using resource data and other user data. As a result, confirmation and retransmission measures are required for the host application when strong ACID (Atomicity, Consistency, Isolation, Durability) properties are needed.

- Non-guarantee of transaction order
  - [E.g.] When receiving consecutive data packets, the order in which data is recorded may differ from the order in which it is received.
  - [E.g.] When events are triggered upon receiving user data, the events may occur in a different order to the order in which the data is received.
- Non-guarantee of transaction completion
  - [E.g.] Received data may not be stored and recorded depending on the timing of when failures occur.

7.1.3 Concurrent Access

- All service APIs are thread-safe and allow for concurrent parallel access.
- However, as mentioned in the previous section, the execution order for tasks is not guaranteed.
- If the same "Resource_JSON" data is registered multiple times at the exact same time all access will be recorded. Access records will not be overwritten.
- If the same "Resource_Binary" data is registered multiple times at the exact same time, only one access record will be kept, and all other data will be overwritten.

7.1.4 Character Codes Used With APIs

Only UTF-8 characters are supported.

7.1.5 Time

Coordinated Universal Time +00:00 (UTC) is used.

7.1.6 Cloud Server Certificates

A server certificate must be deployed to the external server when communicating with an external server via HTTPS using transfer resources and events. Certificates without a public CA signature cannot be used as server

[1] Please note that SSL usage fee is more expensive than API usage fee. Moreover, fee for API usage, SSL usage, and event usage will be charged pay per use basis. We offer a free usage amount for each price plan. Please review your price plan and consider if it should be changed when usage pattern changes greatly.

certificates when doing so.

7.1.7 Data Reachability

Request received responses may be lost after being returned by this service. Data is particularly susceptible to being lost during transfers and event sending, even after attempts to send the data to the transfer destination after success request responses.

7.1.8 Number of Cloud Server API Sessions

A different session is used for each transfer and event sent, regardless of whether the resource is the same. Implement APIs at transfer destinations to create a number of sessions matching the transfer volume.

7.1.9 About REST/MQTT compatibility for the same Resource

In "Resource _ JSON", the resource data registered with REST can be referred by MQTT with MQTT SUBSCRIBE.

Howevver, in "Resource _ Binary", the resource data registered with REST can not be referred by MQTT with MQTT SUBSCRIBE.

7.1.10 About Setting external Server when using Event/Transfer feature

If an API of external server is invoked when using Event/Transfer feature, [Content-Length] needs to be set to the HTTP Response Header of that external Server.

7.1.11 Response for Uses that Exceed Usage Guidelines

Please contact us via our sales department regarding data registration or reference frequency if you might exceed any one of the following guidelines for use concerning the following as we may not be able to prepare a usage environment that suits your needs.

<Guidelines for use>

Single-communication data size		256 Kbytes
Frequency of communications (peak)	Data system API	100 times/second
	Management system API*	1 time/second
No. of registered resources		10,000
Data storage limit within one resource		100 Mbytes
No. of simultaneous connections		100

7.2 REST

7.2.1 Restrictions on Acquired Data Volume

Data cannot be acquired when the acquired data meet any one of the following conditions.

- If there are 1,001 acquired data items or more
- If the total acquired data volume for “Resource_JSON” is larger than 16 MBs
- If the total acquired data volume for “Resource_Binary” is larger than 100 MBs

The amount of data that can be acquired will be written in the body text of the response received when data cannot be acquired due to either of the above conditions being met. Specify the number of data items in the “top” field and publish the API again. Refer to Appendix 1 of the “IoT Platform API Reference” for more details concerning the response body text.

7.3 MQTT

7.3.1 MQTTS when Access Control to Access Code is set

When IP address access control is set to the access code, MQTTS communication is not possible using that access code.

7.3.2 QoS

While MQTT QoS is supported, processing errors within the system after data is received may prevent the arrival of data.

7.3.3 Behavior When a Cut Connection occers

Please be aware that retain data may disappear when the MQTT connection is cut due to server maintenance or for some other reason.

7.3.4 Assigning the retain Flag

A retain flag will retain messages independently for REST and MQTT when set when registering from REST and MQTT.
In this instance, if the MQTT client is set to SUBSCRIBE to the relevant topic, both messages will be published.

7.3.5 KeepAlive

Connections maintained with the MQTT keepalive feature are susceptible to being disconnected due to the communication network based on communication network settings.

7.3.6 Clinet ID at Reconnection

Sometimes, connection might get disconnected during MQTT communication. In such case if reconnecting immediately after getting disconnected, the same Client ID should be used before and after disconnection. Please design MQTT client application considering this.

7.3.7 Other Matters

The current specifications take precedence for all other matters not expressly written in this manual.

7.4 Others

7.4.1 Number of Resources related to a single Access Code

Maximum number of resources that can be related to a single access code is up to 1000.

7.4.2 Others

In case of there is any specification which is not mentioned in this guide, the actual specification of the service will be given priority.

Chapter 8 Controlling Resource Data (REST)

This section provides an example of controlling resource data in this service with an API. Resource data is used by actual command inputs and program execution.

Utilize the technical information provided after service contract. The following is used for the examples provided in this chapter.

Tenant ID	<tenant-id>
Base URI	http://<zone>.fujitsu.com/

8.1 Example of Registering Resource Data

You can register JSON data to “Resource_JSON”.

This example shows the process used to call an API to register attendance information data to the "Attendance Management (Hanako Fujitsu)" resource, and then checking whether this data has actually been transferred.

In this example, attendance information is maintained with the "diligence" resource data variable, with "1" meaning in attendance, and "0" meaning on holiday. Now, we will register that Hanako Fujitsu attended work at 9AM Japanese time on August 3, 2015.

Please create the following resource from the service portal beforehand.

Resource type	Resource
Resource path	diligence/status/hanako
Data format	JSON
Retention period	1 day

Assign the following access code and access permissions for the resource path below beforehand.
You can also register the transfer destination URI in the Service Portal.

Resource path	DiligenceManagement22222
Access code	diligence/status/hanako
Access permission	RU

The R access permission is for reading data, and the U permission is for writing data.

The parameters required to call the data registration API are as follows.

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer DiligenceManagement22222
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako.json?$date=20150803T090000Z
BODY	{ "diligence": 1}

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X PUT -H 'Authorization: Bearer DiligenceManagement22222' -d ' { "diligence": 1 }'
'http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako.json?$date=20150803T090000Z'
```

A "200 OK" response will be returned when the process successfully completes.

8.2 Example of Transferring Resource Data

You can transfer JSON data to the transfer destination using “Resource (Transfer)_JSON”.

This example shows the process used to call an API to register attendance information data to the "Attendance Management (Taro Fujitsu)" resource (transfer), and then checking whether this data has actually been registered.

In this example, attendance information is maintained with the "diligence" resource data variable, with "1" meaning in attendance, and "0" meaning on holiday. Now, we will register that Taro Fujitsu attended work at 9AM Japanese time on August 3, 2015.

Please create the following resources beforehand.

Although for convenience sake, in this example resource of Section 8.1 is being used as the forwarding destination, you may use other servers too. Moreover, if forwarding it only to the client that does MQTT Subscribe, URI need not be specified.

Resource type	Resource(transfer)
Resource path	_fwd/diligence/status/taro
Data format	JSON
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_fwd/diligence/status/hanako
Method	PUT
Header field name	Authorization
Header field value	Bearer DiligenceManagement22222

Next, assign the following access code and access permissions for the resource path below beforehand.

Resource path	DiligenceManagement11111
Access code	_fwd/diligence/status/taro
Access permission	RU

The R access permission is for reading data, and the U permission is for writing data.

The parameters required to call the data registration API are as follows.

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer DiligenceManagement11111
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_fwd/diligence/status/taro.json
BODY	{ "diligence": 1}

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X PUT -H 'Authorization: Bearer DiligenceManagement11111' -d ' { "diligence": 1 }'
'http://<zone>.fujitsu.com/v1/<tenant-id>/_fwd/diligence/status/taro.json'
```

A "200 OK" response will be returned when the process successfully completes.

Memo

Responses for transfers using resources (for transfers) show that transfers from the application publishing the REST have arrived at this service. This does not guarantee arrival at the transfer destination from this service.

8.3 Example of Updating Resource Data

You can update JSON data for “Resource_JSON”.

This example shows the process used to call an API to update attendance information data for the "Attendance Management (Hanako Fujitsu)" resource, and then checking whether the registered data has actually been updated.

You can update the resource data registered in Chapter 8.1 by adding "_past(20150803T090000+0900).json" after the resource path.

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer DiligenceManagement22222
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_past(20150803T090000Z).json
BODY	{ "diligence": 0}

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X PUT -H 'Authorization: Bearer DiligenceManagement22222' -d ' { "diligence": 0 }'
'http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_past(20150803T090000Z).json?$newdate=20150803T100000Z'
```

A "200 OK" response will be returned when the registration process successfully completes.

Memo

Be aware that when multiple resource data entries are registered at the same time, only one resource data entry will be updated.

8.4 Example of Referencing Resource Data

You can reference JSON data in “Resource_JSON”.

This example shows the process used to call an API to reference attendance information data to the "Attendance Management (Hanako Fujitsu)" resource, and then referring to the most recently registered data.

Follow the steps outlined in Chapter 8.1 to first acquire an access code and register data.
The parameters required to call the data reference API are as follows.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer DiligenceManagement22222
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_present.json

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X GET -H 'Authorization: Bearer DiligenceManagement22222'
```

```
'http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_present.json'
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output example easier to understand, the actual output results will not be displayed in this manner.

```
[
  {
    "_data": {
      "diligence": 0
    },
    "_date": "20150803T090000.000Z" ,
    "_resource_path": "diligence/status/hanako"
  }
]
```

JSON data when referencing is configured in the following manner.

Key	Value
_data	Data registered in a format according to the <extension>
_date	Target data registration timestamp

8.5 Example of Retrieving Resource Data

You can retrieve JSON data in “Resource_JSON”.

This example shows the process used to call an API to retrieve attendance information data to the "Attendance Management (Hanako Fujitsu)" resource, and then checking her attendance status for August.

Follow the steps outlined in Chapter 8.1 to first acquire an access code and register data.

The parameters required to call the data reference API are as follows. You can specify a resource data range by using filter condition operators. In this example we will search for an attendance record for August 1 to August 31, 2015.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer DiligenceManagement22222
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_past?$filter=_date gt 20150801T000000Z and _date lt 20150901T000000Z

Run the following to call the API from a Linux OS device using the curl command.
Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", and "=" with "%3d".

```
$curl -i -X GET -H 'Authorization: Bearer DiligenceManagement22222'

'http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_past?$filter=_date%20gt%20150801T000000Z%20and%20_date%20lt%20150901T000000Z'
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output example easier to understand, the actual output results will not be displayed in this manner.

```
[
 {
  "_data" : {
  "diligence" : 1
  },
  "_date" : "20150803T090000Z" ,
  "_resource_path": "diligence/status/hanako"

 }
]
```

8.6 Example of Deleting Resource Data

You can delete JSON data registered to “Resource_JSON”.

In this example we will delete the attendance record for August 1 to August 31, 2015.

Follow the steps outlined in Chapter 8.1 to first acquire an access code and register data.

The parameters required to call the data reference API are as follows.

HTTP method	DELETE
Header field name	Authorization
Header field value	Bearer DiligenceManagement22222
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_past?$filter=_date gt 20150801T000000Z and _date lt 20150901T000000Z

For example, run the following to call the API from a Linux OS device using the curl command.

Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", and "=" with "%3d".

```
$curl -i -X DELETE -H 'Authorization: Bearer DiligenceManagement22222'

'http://<zone>.fujitsu.com/v1/<tenant-id>/diligence/status/hanako/_past?$filter=_date%20gt%2020150801T000000Z%20and%20_date%20lt%2020150901T000000Z'
```

A "200 OK" header is returned as a response when resource data is successfully deleted.

8.7 Example of Bulk Insert Resource Data

Use the Bulk Insert function to register a batch of JSON data in “Resource_JSON”.

This example looks at calling an API used to register temperature and humidity data from a wearable device.

In this example, we will send temperature and humidity data for the device worn by User ID 0001, data that was acquired at minute intervals from 9:00AM to 10:00AM on October 1, 2016. (To explain briefly, the following example only covers the time period from 9:00AM to 9:05AM.)

Create the following resource from portal beforehand.

Resource type	Resource
Resource path	users/0001/temperature
Data format	JSON
Retention period	I day

Assign the following access code and access permissions for the resource path below beforehand.

Resource path	users/0001/temperature
Access Codes	TemperatureAccessCode
Access permission	RU

The R access permission is for reading data, and the U permission is for writing data.

The parameters required to call the data registration API are as follows.

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer TemperatureAccessCode
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/users/0001/temperature.json?**$bulk=single_resource_path**

BODY	[
	{
	"_date" : "20161001T090000Z",
	"_data": {
	"temperature" : 25.0,
	"moisture" : 10.0
	}
	},{
	"_date" : "20161001T090100Z",
	"_data": {
	"temperature" : 25.3,
	"moisture" : 9.0
	}
	},{
	"_date" : "20161001T090200Z",
	"_data": {
	"temperature" : 26.0,
	"moisture" : 12.0
	}
	},{
	"_date" : "20161001T090300Z",
	"_data": {
	"temperature" : 27.0,
	"moisture" : 17.0
	}
	},{
	"_date" : "20161001T090400Z",
	"_data": {
	"temperature" : 29.0,
	"moisture" : 19.0
	}
	}
	]

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

$curl -i -X PUT -H 'Authorization: Bearer TemperatureAccessCode' -d '<above BODY text>' 'http://<zone>.fujitsu.com/v1/<tenant-id>/users/000/temperature 1.json?$bulk= single_resource_path'

A "200 OK" response will be returned when the process successfully completes.

Bulk Insert does not correspond to Event processing. And, the number of JSON that can be stored in the JSON array is up to 1000.

8.8 Example of Referencing Multiple Resource Data Entries with a Single API Operation ($all)

You can reference multiple JSON data entries in "Resource_JSON" with a single API operation by using $all in the query.

This example looks at calling an API used to extract only the information for 9:00AM to 9:30AM on October 1 from the temperature and humidity data on the wearable device.

This example assumes that the following resources are defined in addition to the resources defined in Chapter 8.7, and that temperature and humidity resource data is stored to each.

Create the following resources from service portal beforehand.

Resource type	Resource
Resource path	users/0001/temperature
Data format	JSON
Retention period	I day

Resource type	Resource
Resource path	users/0002/temperature
Data format	JSON
Retention period	I day

Resource type	Resource
Resource path	users/0003/temperature
Data format	JSON
Retention period	I day

Assign the following access code and access permissions for the resource path below beforehand.
And specify the transfer destination from the portal site.

Access Codes	TemperatureAccessCode
The resource set	users
Access permissions for the resource set	PG permissions granted to the above resources

The G access permission is for reading all data under a specific resource, and the P permission is for writing all data under a specific resource. When you use the access code as above, users/0001/temperature, users/0002/temperature, and users/0003/temperature are permitted to read and write.

The parameters required to call the data registration API are as follows.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer TemperatureAccessCode

URI	http://<zone>.fujitsu.com/v1/<tenant-id>/users/**$all**/_past? $filter=_date gt 20161001T090000Z and _date lt 20161001T093000Z
BODY	None

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X GET -H 'Authorization: Bearer TemperatureAccessCode' '
http://<zone>.fujitsu.com/v1/,tenant-id>/users/$all/_past?$filter=_date%20gt%20 20161001T090000
Z %20and%20_date%20lt%2020161001T093000Z
```

A “200 OK" response will be returned, and the following JSON will be returned when the process successfully completes.

```
[
  {
  "_data": {
     "temperature" : 25.0,
     "moisture" : 10.0
   },
  "_date" : "20161001T090000Z",
   "_resource_path": " users/0001/temperature"
  },{
  "_data": {
     "temperature" : 39.0,
     "moisture" : 30.0
   },
  "_date" : "20161001T090000Z",
   "_resource_path": " users/0002/temperature"
  },{
  "_data": {
     "temperature" : 0.0,
     "moisture" : 10.0
     },
  "_date" : "20161001T090000Z",
  "_resource_path": " users/0003/temperature"
 },{
  "_data": {
     "temperature" : 25.3,
"moisture" : 9.0
},
  "_date" : "20161001T090100Z",
"_resource_path": " users/0001/temperature"
 },{
  "_data": {
     "temperature" : 40.1,
     "moisture" : 10.0
   },
  "_date" : "20161001T090100Z",
```

```
    "_resource_path": " users/0002/temperature"
  },{
    "_data": {
      "temperature" : -3.0,
      "moisture" : 10.0
     }
    "_date" : "20161001T090100Z",
      "_resource_path": " users/0003/temperature "
  },{
(The rest has been omitted)
  }
]
```

When using $all, access codes must have read permissions for all resources they are associated with.

8.9 Example of Referencing only a Specific name in Resource Data (select)

You can reference a specific name from multiple names stored in “Resource_JSON”.
This example shows referencing just the temperature data from the data registered in Chapter 8.7.

The parameters required to call the data registration API are as follows.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer TemperatureAccessCode
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/users/0001/ temperature/_past?$select=temperature
BODY	None

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X GET -H 'Authorization: Bearer TemperatureAccessCode' '
http://<zone>.fujitsu.com/v1/<tenant-id>/users/0001/temperature/_past?$select=temperature'
```

A “200 OK" response will be returned, and the following JSON will be returned when the process successfully completes.

```
[
  {
    {
      "_data": {
      "temperature" : 25.0
     },
     "_date" : "20161001T090000Z",
     "_resource_path": " users/0001/temperature"
    },{
    "_data": {
      "temperature" : 25.3
```

```
},
   "_date" : "20161001T090100Z",
"_resource_path": " users/0001/temperature "
 },{
 (The rest has been omitted)
 }
]
```

8.10 Example of registering a csv/txt/binary/gzip file in "Resource_JSON" and referencing the JSON data

In this service, a csv/txt/binary/gzip file can be registered in "Resource_JSON". The registered data can be reffered only in JSON format. This example shows the method of registering and referencing the data of each type of file.

Create the following resources from service portal beforehand.

Resource type	Resource
Resource path	convert
Data format	JSON
Retention period	1 day

Assign the following access code and access permissions for the resource path below beforehand.

Access Codes	ConvertAcc
Resource path	convert
Access permission	RU

Access permission R is for reading data, and U is for writing data.

The parameters required to call the data registration API are as follows.

[1] Registering csv file in "Resource_JSON" and referencing the JSON data

<Registering>

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/ <tenant-id> /convert.csv
BODY	Registering csv file (file.csv) <Detail of file.csv> name,age,tel Tanaka,30,012-345-6789 Suzuki,40,098-765-4321

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X PUT -H 'Authorization: Bearer ConvertAcc' --data-binary @file.csv 'http://<zone>.fujitsu.com/v1/
<tenant-id>/ convert.csv '
```

A "200 OK" response will be returned when the process successfully completes.

<Referencing>

HTTP method	GET
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/convert/_present.json
BODY	None

```
$curl -i -X GET -H 'Authorization: Bearer ConvertAcc' 'http://<zone>.fujitsu.com/v1/<tenant-id>/
convert/_present.json '
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output example easier to understand, the actual output results will not be displayed in this manner.

```
[
  {
    "_data": {
      {"csv":[["name","age","tel"],
        ["Tanaka",30,"012-345-6789"],
        ["Suzuki",40,"098-765-4321"]]
      },
    "_date": "20170901T095721.680Z",
    "_resource_path": " convert"
  }
]
```

[2] Registering txt file in "Resource_JSON" and referencing the JSON data

<Registering>

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/convert.txt

BODY	Registering txt file (file.txt)
	
	＜Detail of file.txt＞
	#include <stdio.h>
	main() {
	printf(" Hello C World!¥¥n ");
	}

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X PUT -H 'Authorization: Bearer ConvertAcc' --data-binary @file.txt 'http://<zone>.fujitsu.com/v1/
<tenant-id>/ convert.txt '
```

A "200 OK" response will be returned when the process successfully completes.

<Referencing>

HTTP method	GET
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/convert/_present.json
BODY	None

```
$curl -i -X GET -H 'Authorization: Bearer ConvertAcc' 'http://<zone>.fujitsu.com/v1/<tenant-id>/
convert/_present.json '
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output example easier to understand, the actual output results will not be displayed in this manner.

```
[
  {
    "_data": {
      {"txt":"#include <stdio.h>¥nmain() {¥nprintf(¥ " Hello CWorld!¥¥n¥ " );¥n}"
      },
    "_date": "20170901T105721.680Z",
    "_resource_path": " convert"
  }
]
```

[3] Registering binary file in "Resource_JSON" and referencing the JSON data

<Registering>

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/convert.bin
BODY	Registering jpg file (koala.jpg)

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X PUT -H 'Authorization: Bearer ConvertAcc' --data-binary    @./koala.jpg
'http://<zone>.fujitsu.com/v1/<tenant-id>/convert.bin'
```

A "200 OK" response will be returned when the process successfully completes.

<Referencing>

HTTP method	GET
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/convert/_present.json

BODY	None

```
$curl -i -X GET -H 'Authorization: Bearer ConvertAcc' 'http://<zone>.fujitsu.com/v1/<tenant-id>/
convert/_present.json '
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output example easier to understand, the actual output results will not be displayed in this manner.

```
[
  {
    "_data": {
      {"bin":" 4OzVEVYVOOAjeOBr+mBuOaKnuOBp+OBjeOBvuOBm+OCk+OAgg0K・・・"
      },
    "_date": "20170901T115721.680Z",
    "_resource_path": " convert"
  }
]
```

[4] Registering gzip file in "Resource_JSON" and referencing the JSON data

<Registering>

HTTP method	PUT
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/convert.json.gz
BODY	Registering gzip file (file.gz)

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X PUT -H 'Authorization: Bearer ConvertAcc' --data-binary    @./file.gz
'http://<zone>.fujitsu.com/v1/ <tenant-id>/convert.json.gz'
```

A "200 OK" response will be returned when the process successfully completes.

<Referencing>

HTTP method	GET
Header field name	Authorization
Header field value	Bearer ConvertAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/convert/_present.json
BODY	None

```
$curl -i -X GET -H 'Authorization: Bearer ConvertAcc' 'http://<zone>.fujitsu.com/v1/<tenant-id>/
convert/_present.json '
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output

example easier to understand, the actual output results will not be displayed in this manner.

```
[
  {
    "_data": {
      {"csv":[["name","age","tel"],
        ["Tanaka",30,"012-345-6789"],
        ["Suzuki",40,"098-765-4321"]]
      },
    "_date": "20170901T095721.680Z",
    "_resource_path": " convert"
  }
]
```

Refer to the API Reference Guide for more details concerning the JSON conversion.

8.11 Example of retrieving with specifying elements position of the array

It is possible to retrieve with specifying the element position of the array. This example shows the method of this type retrieving. Create the following resources from service portal beforehand.

Resource type	Resource
Resource path	Search
Data format	JSON
Retention period	1 day

Assign the following access code and access permissions for the resource path below beforehand.

Access Codes	SearchAcc
Resource path	Search
Access permission	RU

When you finished creating, please register resource data([1]-[4]:JSON data of the array notation).

```
[1] {"data":[["name","age"],["Tanaka",45]]}

[2] {"data":[["name","age"],["Yamaguchi",26]]}

[3] {"data":[["name","age"],["Sato",37]]}

[4] {"data":[["name","age"],["Noji",26]]}
```

Memo

<How to describe the array>

In the above resource data, "age" may actually be located in row-0 and coloumn-1 of the array name "data".

In this case, the location of "age" is defined as follows.

data.0.1

The registered resource data indicates that the first row means labels("name", "age") and the second row means data of labels in JSON array format. In case of retrieving the resource data for people of 26 years age, specify a element position of the array with JSONPath using filter condition operators.

The parameters required to call the data reference API are as follows.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer SearchAcc
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/search/_past?$filter=data.1.1 eq 26'

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X GET -H 'Authorization: Bearer SearchAcc '
'http://<zone>.fujitsu.com/v1/<tenant-id>/search/_past?$filter=data.1.1%20eq%2026'
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output example easier to understand, the actual output results will not be displayed in this manner.

```
[
  {
    "_data": {
      "data": [["name","age"],
        ["Noji",26]]
    },
    "_date": "20170901T095721.680Z",
    "_resource_path": " search"
  },{
    "_data": {
      "data": [["name","age"],
        ["Yamaguchi",26]]
    },
    "_date": "20170901T095647.245Z",
    "_resource_path": " search"
  }
]
```

Chapter 9 Controlling Resource Data (MQTT)

This section introduces examples of using service MQTT APIs. Follow these steps to register and reference resources. Prepare the technical information provided after service contract. The following is used for the examples provided in this chapter. Change the following details as required[2]. The MQTT user name is fixed for each tenant. The MQTT password must be set in the Service Portal.

	Item	Information
1	Tenant ID	<tenant-id>
2	MQTT broker address	<zone>.fujitsu.com
3	IoT base URI	http://<zone>.fujitsu.com/
4	MQTT user name	<tenant-id>

9.1 Example of Registering Resource Data

As with Chapter8.1, this example shows the process used to call an API to register attendance information data to the "Attendance Management (Hanako Fujitsu)" resource, and then referencing whether this data has actually been registered.

In this example, attendance information is maintained with the "diligence" resource data variable, with "1" meaning in attendance, and "0" meaning on holiday. Here we will register that Hanako Fujitsu is in attendance at the current time. Follow the steps outlined in Chapter 8.1 to first acquire an access code.
The parameters required to call the data registration API are as follows.

MQTT broker address	<zone>.fujitsu.com
Port number	1883/TCP (If SSL/TLS, 8883/TCP)
Message type	PUBLISH
MQTT user name	<tenant-id>
MQTT Password	<MQTT password>
Topic	DiligenceManagement22222/v1/<tenant-id>/diligence/status/hanako
Header: DUP flag	0
Header: QoS flag	0
Header: RETAIN flag	True

Payload	{
	"diligence": 1
	}

The source code to run on Python with the Eclipse Paho library is shown below. Set the file name to pub.py.

```
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc):
    if str(rc)=='0':
        print('[CONNACK]:client can connect :' + str(rc));
    else:
        print('[CONNACK]:client can¥'t connect :' + str(rc));
    client.publish(topic,payload,qos=0,retain=False)
```

[2] With MQTT, you can register and reference data resources, but you cannot update, search or delete them. Use REST if you prefer to update, search or delete data resources.

```
def on_publish(client,userdata,mid):
    print("client is published ")
    client.disconnect()

if __name__ == '__main__':
    is_connect = True

    username = ' <tenant-id'
    password = '<MQTT Password>'
    host = '<zone>.fujitsu.com'
    topic = 'DiligenceManagement22222/v1/<tenant-id>/diligence/status/hanako'
    port = 1883
    payload = '{"diligence":1}'
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_publish = on_publish
    client.username_pw_set(username,password)
    client.connect(host,port,keepalive=1800)
    client.loop_forever()
```

The source code to run on Python with the Eclipse Paho library is shown below. Run the following.

```
$python pub.py
```

The following will be output.

```
[CONNACK]:client can connect :0
client is published
```

With MQTT, a normal response will be given even when the JSON data format is faulty, making it impossible to determine whether data storage on this service was successful or not. Reference the resource data in REST to determine whether it has been registered properly.

9.2 Example of Referencing Resource Data

This example covers using MQTT to reference registered resource data. Follow the steps outlined in Chapter 8.1 to first acquire an access code. The parameters required to call the data registration API are as follows.

MQTT broker address	<zone>.fujitsu.com
Port number	1883/TCP (If SSL/TLS, 8883/TCP)
Message type	SUBSCRIBE
Topic	DiligenceManagement22222/v1/<tenant-id>/diligence/status/hanako
Keep-alive	1800 seconds
Header: DUP flag	0
Header: QoS flag	0
Header: RETAIN flag	False
Payload	DiligenceManagement22222/v1/<tenant-id>/diligence/status/hanako

Set the file name to sub.py.

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    if str(rc)=='0':
        print('[CONNACK]:client can connect :' + str(rc));
    else:
        print('[CONNACK]:client can¥'t connect :' + str(rc));
    client.subscribe(topic,qos=0)

def on_message(client, userdata, msg):
    print('Message is recieved.¥r¥nTOPIC:' + msg.topic + '¥r¥npayload:¥r¥n' + str(msg.payload))
    client.disconnect();

if __name__ == '__main__':
    username = '<tenant-id>'
    password = '< MQTTpasword>'
    host = '<zone>.fujitsu.com'
    topic='DiligenceManagement22222/v1/<tenant-id>/diligence/status/hanako'
    port = 1883

    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message

    client.username_pw_set(username, password)
    client.connect(host, port=port, keepalive=1800)
    client.loop_forever()
```

Run the following.

```
$python sub.py
```

The following process created in Chapter 9.1 will be run from a different terminal.

```
$python pub.py
```

After running this, the following will be returned when registering data.

```
topic: DiligenceManagement22222/v1/<tenant-id>/diligence/status/hanako

payload:
{ "diligence" : 1}
```

Memo

There are limits to how much CONNECT, SUBSCRIBE status can be maintained.
The KeepAlive value that can be specified when in the CONNECT status is from 1 to 1800 seconds.
As a result, the system may not be able to maintain a session depending on the data communication status. Either maintain the session using PINGREQ, or try reconnecting after disconnecting.
(You can receive information later when disconnecting if retain is set to True when publishing.)
*Connections may be rejected if repeated connection attempts are made using CONNECT only.

Chapter 10 Controlling Resource_Binary Data

This section provides an example of controlling resource data in Resource_Binary in this service with an API. Resource data is used by actual command inputs and program execution.
This example involves capturing regular footage of line A and line B for the "Tokyo Kamata Plant", and confirming that this data can be registered, retrieved, referenced and deleted with this service.

Prepare the technical information provided after service contract. The following is used for the examples provided in this chapter. Change the following details as required.

Tenant ID	<tenant-id>
Base URI	http://<zone>.fujitsu.com/

Create resources and access codes from the Service Portal in advance as shown below.
Refer to the "IoT Platform Service Portal Operating Manual" for more information on how these are created.

Resource type	Resource
Resource path	_bin/Tokyo/Kamata/LineA/normal
Data format	Binary
Retention period	1 day

Resource path	_bin/Tokyo/Kamata/LineA/normal
Access Codes	AccTokyoLineAllRU
Access permission	Select "R" and "U"

The "R" access permission is for reading resource data, and the "U" permission is for writing resource data.

10.1 Example of Registering Resource_Binary Data

You can register Binary data to "Resource_Binary".

The example used here involves registering images of manufacturing line A in normal operation at the Tokyo Kamata Plant. Specifically, this consists of registering binary images to Resource_Binary "_bin/Tokyo/Kamata/LineA/normal".

In this example, images are compressed and managed in the JPG file format. This example assumes that we are registering an image (lineA1000.jpg) showing that manufacturing line A is operating normally as of 10:01AM September 1, 2016 JST. This example also assumes that the "productID12345" character string is registered as metadata for the image.

The parameters required to call the data registration API are as follows.

HTTP method	PUT
Header field name 1	Authorization
Header field value 1	Bearer AccTokyoLineAllRU
Header field name 2	x-iotpf-meta-data1
Header field value 2	productID12345
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal?$date=20160901T100100Z
BODY	The JPEG file registered (lineA1000.jpg)

For example, run the following to call the API from a Linux OS device using the curl command. Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", "'" (single quotation mark) with "%27", and "=" with "%3d".

```
$curl -i -X PUT -H 'Authorization: Bearer AccTokyoLineAllRU' -H ' x-iotpf-meta-data1: productID12345' --data-binary @lineA1000.jpg 'http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal?$date=20160901T100100Z'
```

A "200 OK" response will be returned when the process successfully completes.

10.2 Example of Retrieving Resource_Binary Data

The example used here involves retrieving images of manufacturing line A in normal operation at the Tokyo Kamata Plant. Specifically, this consists of retrieving binary images from Resource_Binary "_bin/Tokyo/Kamata/LineA/normal".

Follow the steps outlined in Chapter 10.1 to first acquire an access code and register data.

In this example, images are compressed and managed in the JPG file format. This example assumes that we are retrieving images showing that manufacturing line A is operating normally from 10:00AM to 11:00AM on September 1, 2016 JST. Here, we assume that data has been registered for 10:01AM, 10:21AM, and 10:41AM in advance.

The parameters required to call the data registration API are as follows. You can specify a resource data range by using filter condition operators. In this example, we will retrieve the attendance record for September 1, 2016 from 10:00AM to 11:00AM JST. Refer to the "IoT Platform API Reference" for more information about filter condition operators.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer AccTokyoLineAllRU
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal/_past?$filter =_date gt 20160901T100000Z and _date lt 20160901T110000Z

Run the following to call the API from a Linux OS device using the curl command.
Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace " " (single-byte space) with "%20", "+" with "%2b", and "=" with "%3d".

```
$curl -i -X GET -H 'Authorization: Bearer AccTokyoLineAllRU
 'http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal
/_past?$filter=_date%20gt%2020160901T100000Z%20and%20_date%20lt%2020160901T110000Z
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced. While line breaks have been added where appropriate to make the following output example easier to understand, the actual output results will not be displayed in this manner.

```
[
{
"content_type" : "application/x-www-form-urlencoded",
"name" : "_bin/Tokyo/Kamata/LineA/normal/_past(20160901T100100.000Z)"
},
{
"content_type" : "application/x-www-form-urlencoded",
"name" : "_bin/Tokyo/Kamata/LineA/normal/_past(20160901T102100.000Z)"
},
{
"content_type" : "application/x-www-form-urlencoded",
"name" : "_bin/Tokyo/Kamata/LineA/normal/_past(20160901T104100.000Z)"
```

```
}
]
```

When searching for resource data in the Resource_Binary, the actual binary data is not included in the response. To acquire the actual binary data, you will need to reference and acquire each piece of binary data one at a time. (Refer to Chapter 10.3)

10.3 Example of Referencing Resource_Binary Data

This example involves referencing images of manufacturing line A in normal operation at the Tokyo Kamata Plant. Specifically, this consists of referencing images registered to Resource_Binary "_bin/Tokyo/Kamata/LineA/normal" on September 1, 2016 at 10:01AM JST.

Follow the steps outlined in Chapter 10.1 to first acquire an access code and register data.

The parameters required to call the data reference API are as follows.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer AccTokyoLineAllRU
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal/_past(20160901T100100.000Z)

For example, run the following to call the API from a Linux OS device using the curl command.
Characters other than non-reserved characters must be replaced with a percentage encoding from the URI $filter= string on. Replace "+" with "%2b".

```
$curl -X GET -H 'Authorization: Bearer AccTokyoLineAllRU'
 'http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal/_past(20160901T100100.000Z  )'
> outputLineA.jpg
```

The binary data for the set time (in this example, the JPG file registered in Chapter10.1) is returned as a response along with "200 OK" as the header when resource data is successfully referenced.

Memo

Note that communications may be disconnected when attempting to reference a large amount of binary data. If this happens, use HTTP Range Request to download the remaining data after disconnection.

Reference the Content-Range for the Header if 206 Partial Content is returned. Next, specify the Range for the Header and call the API to acquire the remaining data.

10.4 Example of Deleting Resource_Binary Data

This example involves deleting images of manufacturing line A in normal operation at the Tokyo Kamata Plant. Specifically, this consists of deleting images registered to Resource_Binary "_bin/Tokyo/Kamata/LineA/normal" on September 1, 2016 at 10:01AM JST.

Follow the steps outlined in Chapter 10.1 to first acquire an access code and register data.

The parameters required to call the data reference API are as follows.

HTTP method	DELETE

Header field name	Authorization
Header field value	Bearer AccTokyoLineAllRU
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal/_past?$filter=_date eq 20160901T100100.000Z

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X DELETE -H 'Authorization: Bearer AccTokyoLineAllRU'

'http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineA/normal/_past?$filter=_date%20eq%20160901T100100.000Z'
```

A "200 OK" header is returned as a response when resource data is successfully deleted.

Chapter 11 Resource Control

This section provides an example of controlling resources in this service with an API. Resources are used by actual command inputs and program execution.
This example involves registering, referencing, and deleting the resources, access codes and events needed in advance to capture regular footage of line A and line B for the "Tokyo Kamata Plant" introduced in Chapter 10.

Prepare the technical information provided after service contract. The following is used for the examples provided in this chapter. Change the following details as required.

Tenant ID	<tenant-id>
Base URI	http://<zone>.fujitsu.com/

Create resources and access codes from the Service Portal in advance as shown below.
Refer to the "IoT Platform Service Portal Operating Manual" for more details on creating resources and access codes.

Resource type	Resource
Resource path	_bin/Tokyo
Data format	Binary
Retention period	1 day

Assign the following access code and access permissions for the resource path below beforehand.
And specify the transfer destination from the portal site.

Resource path	_bin/Tokyo
Access Codes	AccTokyoCDL
Access permission	Select "CDL"

"CDL" access permissions mean that users can create (C), delete (D) and reference (L) resources with an API for resources under the resource path. Creating the resources and access codes mentioned above beforehand makes it possible to use an API to create and delete resources in the resource path under "_bin/Tokyo".

This negates the need to use the Service Portal each time the number of devices being used increases.

11.1 Example of Registering Resources

The example used here involves creating a resource for registering images of manufacturing line B at the Tokyo Kamata Plant. Specifically, this consists of registering Resource_Binary "_bin/Tokyo/Kamata/LineB/normal" via an API. The resource registered is as follows.

Resource type	Resource_Binary
Resource path	_bin/Tokyo/Kamata/LineB/normal
Data format	Binary
Retention period	3 days

The parameters required to call the API are as follows.

HTTP method	POST
Header field name	Authorization
Header field value	Bearer AccTokyoCDL

URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineB/normal

BODY	{
	"resource" : {
	“retention_period” : 3
	}
	}

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X POST -H 'Authorization: Bearer AccTokyoCDL' -d '{"resource" : {
 "retention_period" : 3}}'          'http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineB/normal'
```

A "201 Created" response will be returned when the process successfully completes. Additionally, the resource path registered as the Location (http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineB/normal) will be returned as the header.

11.2 Example of Referencing Resource Metadata

This example involves referencing all resources registered under “_bin/Tokyo”. Refer to Chapter 11.1 to register these resources in advance. The parameters required to call the API are as follows.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer AccTokyoCDL
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/$all/_resources

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X GET -H 'Authorization: Bearer AccTokyoCDL'
 'http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/$all/_resources'
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced.

```
{
 "resources" : [{
  "resource_path" : "_bin/Tokyo/Kamata/LineA/normal",
  "retention_period" : 1,
  "last_modified" : 20160901T100100Z
  }
 },
 {
  "resource_path" : "_bin/Tokyo/Kamata/LineA/fault",
  "retention_period" : 1,
  "last_modified" : 20160901T100000Z
  }
  },
  {
   "resource_path" : "_bin/Tokyo/Kamata/LineB/normal",
   "retention_period" : 3,
```

```
    "last_modified" : 20160901T120100Z
   }
  },{
   "resource_path" : "_bin/Tokyo/Kamata/LineB/fault",
   "retention_period" : 1,
   "last_modified" : 20160901T120000Z
   }
  }
]
```

In this example, confirm that four resources, "_bin/Tokyo/Kamata/LineA/normal", "_bin/Tokyo/Kamata/LineA/fault", "_bin/Tokyo/Kamata/LineB/normal", and "_bin/Tokyo/Kamata/LineB/fault" are registered under "_bin/Tokyo".

11.3 Example of Deleting Resources

The example used here involves deleting a resource (_bin/Tokyo/Kamata/LineC/normal) for registering images of manufacturing line C at the Tokyo Kamata Plant. Refer to Chapter 11.1 to register these resources in advance.

Resource type	Resource_Binary
Resource path	_bin/Tokyo/Kamata/LineC/normal
Retention period	3 days

The parameters required to call the data reference API are as follows.

HTTP method	DELETE
Header field name	Authorization
Header field value	Bearer AccTokyoCDL
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineC/normal

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X DELETE -H 'Authorization: Bearer AccTokyoCDL '
 'http://<zone>.fujitsu.com/v1/<tenant-id>/_bin/Tokyo/Kamata/LineC/normal'
```

A "204 No Content" header is returned as a response when the resource is successfully deleted.
A "423 Locked" is returned as a response when access code is set to the resource. In such case, access code needs to be deleted before deleting the resource. Please refer to Chapter 12.3 for deleting access code.

Chapter 12 Controlling Access Codes

12.1 Example of Registering Access Codes

Register the access code for the resource registered in Chapter 11.1. Use this to register resource data using the access code registered. The access code registered is as follows.

Access Code	AccTokyoLineAllRU
Resource path 1	_bin/Tokyo/Kamata/LineB/normal
Access permission	“R” and “U”

The “R” (read) access permission is for reading resource data, and the “U” (update) permission is for writing resource data.

HTTP method	POST
Header field name	Authorization
Header field value	Bearer AccTokyoCDL
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_access_code/AccTokyoLineAllRU

BODY	{
	"access_code": {
	"permissions": {
	"resource_operations": [
	{
	"operations": [
	"update",
	"read"
	],
	"resource_path": "_bin/Tokyo/Kamata/LineB/normal"
	}
	]
	}
	}
	}

For example, run the following to call the API from a Linux OS device using the curl command.

```
$ curl -i -X POST -H 'Authorization: Bearer AccTokyoCDL' -H 'Content-Type: application/json; charset=UTF-8' -d
'{"access_code" :{"permissions": {"resource_operations":[{"resource_path":
"_bin/Tokyo/Kamata/LineB/normal","operations":["update","read"]}]}}}'
'http://<zone>.fujitsu.com/v1/<tenant-id>/_access_codes/AccTokyoLineAllRU'
```

A "201 Created" response will be returned when the process successfully completes. Additionally, the access code registered as the Location (http://<zone>.fujitsu.com/v1/<tenant-id>/_access_code/AccTokyoLineAllRU) will be returned as the header.

12.2 Example of Referencing Access Codes

This example involves referencing detailed information (the resource path and resource path permissions) for the access code registered in Chapter 12.1. Refer to Chapter 11 and Chapter 12.1 to register the access code.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer AccTokyoCDL
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_access_code/AccTokyoLineAllRU

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X GET -H 'Authorization: Bearer AccTokyoCDL'              '
http://<zone>.fujitsu.com/v1/<tenant-id>/_access_code/AccTokyoLineAllRU'
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced.

```
{
  "access_codes" : [ {
    "permissions" : {
      "ip_filter" : [ ],
      "resource_operations" : [ {
        "operations" : [ "read", "update" ],
        "resource_path" : "_bin/Tokyo/Kamata/LineA/normal"
      } ]
    },
    "access_code" : "AccTokyoLineAllRU"
  } ]
}
```

12.3 Example of Deleting Access Codes

This example covers deleting the access code registered in Chapter 12.1. Refer to Chapter 12.1 to register these resources in advance.

Access Code	AccTokyoLineAllRU
Resource path	_bin/Tokyo/Kamata/LineB/normal
Access permission	“R” and “U”

The parameters required to call the data reference API are as follows.

HTTP method	DELETE
Header field name	Authorization
Header field value	Bearer AccTokyoCDL
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_access_code/AccTokyoLineAllRU

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X DELETE -H 'Authorization: Bearer AccTokyoCDL '
 'http://<zone>.fujitsu.com/v1/<tenant-id>/_access_code/AccTokyoLineAllRU'
```

A "204 No Content" header is returned as a response when the resource is successfully deleted.

Chapter 13 Controlling Events

13.1 Example of Registering Events

In this example, we create an event to send an email whenever the temperature sensor at site A exceeds 30 degrees Celsius. Create the following resource and access code in advance.

Resource type	Resources
Resource path	sensors
Data format	JSON
Retention period	1 day

Resource type	Resources
Resource path	sensors/temperature
Data format	JSON
Retention period	1 day

Access code	TemperatureCDL
Resource path 1	sensors
Access permission	“CDL”

Access code	TemperatureRU
Resource path 1	sensors/temperature
Access permission	“R” and “U”

“CDL” access permissions mean that users can create (C), delete (D) and reference (L) resources with an API for resources under the resource path.

The “R” (read) access permission is for reading resource data, and the “U” (update) permission is for writing resource data.

HTTP method	POST
Header field name1	Authorization
Header field value1	Bearer TemperatureCDL
Header field name2	Content-Type
Header field value2	application/json; charset=UTF-8
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_events

BODY	{
	"event" : {
	"notification" : {
	"smtp" : {
	"subject" : "Temperature exceeded 30 degrees Celcius",
	"body" : "The temperature has exceeded 30 degrees Celcius. Please confirm.”,
	"send_to" : "hoge@piyo.fujitsu.com"
	}
	},

	"conditions" : {
	"targets" : {
	"read_access_code" : "TemperatureRU",
	"operations" : ["create","update"],
	"resource_path" : "sensors/temperature"
	},
	"notification_condition" : {
	"body_conditions" : {
	"path" : "sensors/temperature",
	"path_type" : "JSONPath",
	"comparing_operator" : "ge",
	"value" : 30.00000
	}
	}
	}
	}}

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X POST -H 'Authorization: Bearer TemperatureCDL' -H 'Content-Type : application/json; charset=UTF-8' -d '<the BODY part above>' ' http://<zone>.fujitsu.com/v1/<tenant-id>/_events'
```

A "201 Created" response will be returned when the process successfully completes. Additionally, the event number for the event registered as the Location (http://<zone>.fujitsu.com/v1/<tenant-id>/_events/1234567890) will be returned as the header. The event ID will be 1234567890, so keep a record of this.

13.2 Example of Referencing Events

This example involves referencing the event registered in Chapter 13.1. Refer to Chapter 13.1 to register the event in advance.
In this example, the event ID will be “1234567890”.

HTTP method	GET
Header field name	Authorization
Header field value	Bearer TemperatureCDL
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_events/1234567890

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X GET -H 'Authorization: Bearer TemperatureCDL'
'http://<zone>.fujitsu.com/v1/<tenant-id>/_events/1234567890'
```

The following JSON data is returned as a response along with "200 OK" as the header when resource data is successfully referenced.

```
{
  "events" : [ {
    "conditions" : {
      "targets" : {
        "operations" : [ "create", "update" ],
```

```
            "resource_path" : "sensors/temperature",
            "read_access_code" : "TemperatureRU"
         },
         "notification_condition" : {
            "body_conditions" : {
               "path" : "sensors/temperature",
               "path_type" : "JSONPath",
               "comparing_operator" : "ge",
               "value" : 30.00000
            }
         }
      },
      "notification" : {
         "smtp" : {
            "subject" : "Temperature exceeded 30 degrees Celcius",
            "body" : "The temperature has exceeded 30 degrees Celcius. Please confirm.",
            "send_to" : " hoge@piyo.fujitsu.com"
         }
      },
      "event_id" : "1234567890"
   } ]
}
```

13.3 Example of Deleting Events

This example involves deleting the event registered in Chapter 13.1. Refer to Chapter 13.1 to delete the event in advance.
The parameters required to call the delete event API are as follows.

HTTP method	DELETE
Header field name	Authorization
Header field value	Bearer TemperatureCDL
URI	http://<zone>.fujitsu.com/v1/<tenant-id>/_events/1234567890

For example, run the following to call the API from a Linux OS device using the curl command.

```
$curl -i -X DELETE -H 'Authorization: Bearer TemperatureCDL '
  'http://<zone>.fujitsu.com/v1/<tenant-id>/_events/1234567890'
```

A "204 No Content" header is returned as a response when the resource is successfully deleted.

Appendix 1 REST App (Android)

A set of sample Android app project tools is available to use with resource data and to acquire logs.
Inquire with the sales representative for the means of acquisition for these files.

Licenses

The following licenses apply in relation to this app.
Please read these license conditions thoroughly before using this app.

Operating Environment

Unzip and import "SampleAppREST.zip" into Android Studio for use. Refer to the official Android Developer website for more information about importing files and build procedure information.

This app has been confirmed to run on the following operating environment.

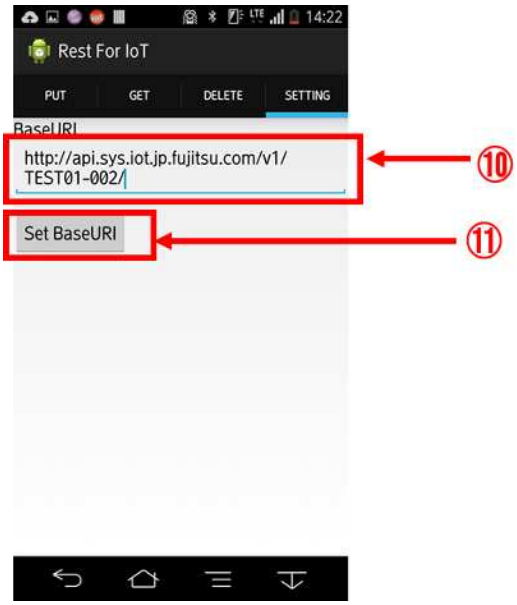
- ARROWS NX F-05F (Android 4.4)

Additionally, the API level running the application is as follows. The same SDK as the target build SDK version is required when building the application.

	API level	Android version
Minimum SDK version	19	Android 4.4 KitKat
Target build SDK version	21	Android 5.0 Lollipop

Overview

	Item	Overview
①	Method tab	Tap to select the method you prefer to run from PUT/GET/DELETE. You can also configure the base URI from the SETTING tab.
②	Base URI	Set to the base URI configured in the SETTING tab.
③	Resource path	The resource path and query set as a character string. A percentage mark encoded character string is required for a query. The resource path input when the Run REST button is pressed is loaded in.
④	Access code	The access code set as a character string. The access code input when the Run REST button is pressed is loaded in.
⑤	BODY text	The BODY text set as a character string. The BODY text input when the Run REST button is pressed is loaded in.
⑥	Issue REST button	Runs REST with the content set in ① to ⑤ when pressed. Execution results are displayed in the response display section.
⑦	Clear button	Clears response display section content.
⑧	Toast	Displays the output destination for execution logs. This toast will only appear for a set period of time after the Run REST button is pressed. Execution logs show REST content and responses when executed in a folder on the Android device. A single file is output each time an API is run. Files are stored in the following location. /storage/emulated/0/Download
⑨	Response display section	Displays the results of executing the process with the Issue REST button.



	Item	Overview
⑩	Base URI settings	The base URI set as a character string.
⑪	Set Base URI button	The character string input into the base URI settings is applied when pressed. This is reflected in all PUT/GET/DELETE tabs.