PaygOps
Last-Mile Management System

OpenPAYGO Metrics

DRAFT SPECIFICATIONS - v0.11

Purpose

The purpose of this API specification is to provide a standardized way for PAYGO devices to send usage metrics to a server and get back information about their activation status. It is particularly optimized for transmitting data over slow connections and connections that have high costs while maintaining simplicity.

Having this specification open-source allows PAYG device manufacturers to implement the specified behaviour into their devices and let the distributor or final user choose which software to use. The specification also describes how to exchange data between servers, allowing for example that a manufacturer hosts their own server to gather device data but still allow a software provider to access it.

An example use case would be a device that has a budget of 750KB of data per month over a 2G network with sessions rounded to the nearest 1KB. Using that specification the device can make one request per hour in which it transmits 30 updates of each of 5 metrics (battery voltage, battery current, panel voltage, output 1 current, output 2 current) as well as a few alarms and gets back an OpenPAYGO Token securely updating its activation status. This example request including all data fits in less than 1KB including all headers and the response meaning that doing it every hour for the whole month is still below the 750KB monthly data budget while providing a 2min data resolution on 5 key metrics.

Why not use existing standards (such as the OCF one)? Those standards have been developed in another context and while some of them can be adapted to be used in the PAYGO world (see for example those used in the OpenPAYGO Link project for accessory to device communication) the existing Device-to-Cloud standards are poorly optimized for using 2G networks with high costs of services or for running on extremely constrained devices. We welcome suggestions of existing standards that could meet the target use cases while not increasing costs or power consumption and would be happy to retire or adapt this specification if such standards were found.

Getting Started

Simple vs Condensed

The data format is flexible and allows the data to be sent in a format that is either very explicit and easier to read, or a format that is condensed and more efficient in terms of data. Both forms can be mixed and it is easy to convert from one form to the other.

We generally recommend using the simple form for Server to Server communication as well as on the early stage of development of Device to Server communication to make debugging and changes easier. You can then easily switch to the condensed form to gain efficiency.

The main difference is that in the condensed format shorthands are used for the standard variables (e.g. “sn” instead of “serial_number”), and that the data format (e.g. variable names, units, order of variables, etc.) is pre-registered with the server so that only the data itself can be sent with minimal overhead. The condensed format resulted in an average reduction of ~80% in data consumption in our surveyed use cases (not using GZIP or other compression methods).

Simple Example

POST https://example.server.com/examplesubroute/device_data

```
{
   "serial_number":"A111222",
   "timestamp":1611583070,
   "data":{
      "token_count":13,
      "tampered":false,
      "firmware_version":"1.14.2"
   },
   "historical_data":[
      {
         "timestamp":1611583070,
         "panel_voltage":17.5,
         "battery_voltage":12.5,
         "panel_current":2.2,
         "battery_current":3.2
      },
      {
         "timestamp":1611583010,
         "panel_voltage":15.7,
         "battery_voltage":12.6,
         "panel_current":2.2,
         "battery_current":3.2,
         "usb_load_1_current":0.7
      }
   ],
   "data_format":{
      "variables":{
         "battery_current":{
            "name":"Battery Current",
            "type":"float",
            "unit":"A",
            "description":"The battery current, negative means it is charging."
         }
      }
   }
}
```

This is a basic example of posting device data to a server. Note that the data_format is optional.

Condensed Example

POST https://example.server.com/examplesubroute/device_data

```
{
   "sn":"A111222",
   "df":12,
   "d":[13,0,"1.14.2"],
   "hd":[
      [17.5,12.5,2.2,3.2],
      [15.7,12.6,2.2,3.2,0.7]
   ]
}
```

Although this condensed example is used to post the same data as the previous one, we can see that it is made a lot more compact by using a few key features:

1. The data format is predefined, including the orders in which the variables are expected, allowing to store them in lists without names. The data format is then referenced in the "data_format" variable (or "df" for short) with the ID returned by the server when registering it (see example below).

2. The main timestamp is omitted (using the timestamp of the request itself), and the timestamps of the historical data are also omitted by defining a "historical_data_interval" for the metrics of 60 seconds.

3. Shorter versions of the standard variable names are used (e.g. "hd" for "historical_data")

For this condensed example to be accepted, the data format needs to have been previously defined on the server. Usually you would need to do that just once per model of device. We used the example request below to do it:

POST https://example.server.com/examplesubroute/data_format

```
{
   "data_order":[
      "token_count",
      "tampered",
      "firmware_version"
   ],
   "historical_data_interval":-60,
   "historical_data_order":[
      "panel_voltage",
      "battery_voltage",
      "panel_current",
      "battery_current",
```

```
      "usb_load_1_current",
      "usb_load_2_current",
      "overload_alert",
      "timestamp"
   ],
   "variables":{
      "battery_current":{
         "name":"Battery Current",
         "type":"float",
         "unit":"A",
         "description":"The current coming out of the battery, negative means it is charging."
      }
   }
}
```

To this request, the server returned a code **200** with the data **{"id": 12}** allowing us to use the data format with this ID in future requests. **WARNING: Variable keys should NOT be integers even when represented as strings to avoid clashing with potential indexes of defined data format.**

Data Structures

Main Object

Description: This is the main object when sending Device Data, it contains information about the device current state and historical data. It can also optionally contain information about the data format, such as detailed descriptions of the variables.

{

serial_number* — string
example: A111222
short version: sn
Description: This is the serial number (also called PAYGO ID) of the device for which the data is sent.

data** — object
short version: d
Description: This is an object containing current data about the device (e.g. firmware version, tamper status, enabled loads, etc.). For OpenPAYGO Token devices, the data section is expected to contain the "token_count" variable. All other variables are optional

historical_data** — array (of device data objects or device data arrays)
short version: hd

Description: This is an array containing a list of either device data objects or device data arrays (see structures below) in order from oldest to newest. The list can contain a mix of both device data objects and device data structures. Each device data objects or device data array must contain a “timestamp” or “relative_time” unless a “historical_data_interval” has been defined in the data format. If a “historical_data_interval” is defined but an object still provides the time, the time provided by the object overrides the time generated by the interval from the last object.

timestamp — string (dateTime)
short version: ts
example: 2018-06-30T23:58:59Z
Description: This is the timestamp of the request. It is useful if you use “relative_time” in the historical data objects, the relative time is then computed relative to the provided timestamp. If the timestamp is not provided, it is assumed to be the timestamp at the time the request is received.

data_format — object
short version: dfo
Description: This is the data format object describing the data format of the request.

data_format_id — integer
short version: df
Description: The ID of the data_format used in the request.

accessories — array (of main objects)
Description: This allows you to next into one object Main Objects for the accessories connected to that device (e.g. a TV, a Fridge, etc.) that each have their own serial numbers and data format.

}

Notes:

- **At least one of “data” or “historical_data” must be provided for the request to be valid.
- To be able to use device data arrays in the historical_data or data objects, either a “data_format” or a “data_format_id” field must be provided corresponding to a data format that defines variable orders. They cannot be both provided together.

Example (see other examples above):

```
{
   "sn":"A111222",
   "df":12,
   "d":[13,0,"1.14.2"],
```

```
   "hd":[
      [17.5,12.5,2.2,3.2],
      [15.7,12.6,2.2,3.2,0.7],
      {"7":1611583055,"6":1},
      [15.7,12.6,2.2,3.2,0.8],
   ]
}
```

Device Data Object

Description: The device data object is an object containing any number of variables and representing a given state of the device. Those variables can be defined with further details in the data format object (in the "variable" sub-object). Some variables are standard variables, in particular the "timestamp" variable (representing the Unix timestamp at which the data was taken) or the "relative_time" variable (representing a time elapsed in seconds from the time in the reference timestamp provided in the main object, or the time of the request).

Example:

```
{
     "timestamp":1611583010,
     "panel_voltage":17.5,
     "battery_voltage":12.5,
     "panel_current":2.2,
     "battery_current":3.2
}
```

If the Device Data Format defines an object order, then the object order can be used as the key for each of the variable to save data:

Example:

```
{
     "7":1611583010,
     "0":17.5,
     "1":12.5,
     "2":2.2,
     "3":3.2
}
```

Device Data Array

Description: The device data array is an array containing any number of variables and representing a given state of the device. The order of those variables MUST be defined in the data format object (in the "data_order" or "historical_data_order" objects).

Example:

```
[17.5,12.5,2.2,3.2,0,0,0,1611583070]
```

There is no minimum number of variables to be included and variables that are not useful can be omitted as long as they are after the last variable that is needed in the order. Here we omit the usb_load_current 1 and 2 as well as the timestamp that is not needed since the interval has been defined at 60 seconds already).

Example:

```
[17.5,12.5,2.2,3.2]
```

If you want to submit just a few variables that are higher in the order than variables that are not needed (e.g. only sending the "overload_alert" from our example), it is often best to use a device data object instead. (e.g. **{"6":1}** instead of **[0,0,0,0,0,0,1]**, note that the indexes start at 0)

Data Format Object

Description: This is the object used to define the data format of the main object. It can either be included in the main object in the "data_format" sub-object, or can be registered separately and then referenced using the "data_format_id" obtained after registering it (see below the route for registering).

{

data_order	object example: {"1":"token_count", "2":"firmware_version"} Description: This object describes the order of the "data" attribute of the main object if such data is provided as a Device Data Array.
historical_data_order	object example: {"1":"panel_voltage", "2":"battery_voltage"} Description: This object describes the order of the "historical_data" attribute of the main object if such data is provided as a Device Data Array.

historical_data_interval	integer (seconds) example: 300 Description: This defines the default interval in seconds between historical data objects. When using this, it is not needed to specify the "timestamp" or "relative_time" of each of the Device Data Object or Device Data Array in the "historical_data" array. Instead, if the value is not there, it is assumed to be that many seconds after the last item of the array. It can be negative if the objects the highest in the list are the most recent.
variables	object Description: This is the object giving additional details about each of the variables. While this is not mandatory, it is recommended as it provides additional information for the software platform to properly describe and display data coming from the device to their users. The keys of the object are the name of the variables and for each key, there is a sub-object representing the variable. Only the "**name**" is required but you can also include a "**type**" (integer, float, text, bool), a "**unit**" and a "**description**". If not provided, the type is assumed to be "text". A **scale factor** can also be used as a multiplier for the given value.

}

Example:

```
{
   "data_order":[
      "token_count",
      "tampered",
      "firmware_version"
   ],
   "historical_data_interval":-60,
   "historical_data_order":[
      "panel_voltage",
      "battery_voltage",
      "panel_current",
      "battery_current",
      "usb_load_1_current",
      "usb_load_2_current",
      "overload_alert",
      "relative_time"
   ],
   "variables":{
      "battery_current":{
         "name":"Battery Current",
         "type":"float",
```

```
            "unit":"A",
            "description":"The current coming out of the battery, negative means it is charging."
         },
         "overload_alert":{
            "name":"Too Many Loads Connected"
         },
      }
}
```

Routes

Send Device Data

Method: POST

Route: /device_data or /dd

Description: You can use this route to send a collection of device data to a server.

Expected Content: A Main Object matching the Main Object Data Structure.

Expected Response: A 201 code with either:

1. Default: An empty object

2. If “token_count” is provided in the data: An object containing a “token_list” (or “tkl”) key with an array of OpenPAYGO Token to be used by the device from oldest to most recent.

```
{"tkl": [111222333, 333444555, 555666777]}
```

Note that for devices that use “time” and not “credit”, which is most of the devices, usually there is only one “SET_TIME” token corresponding to the difference between now and the target expiration time of the device.

3. If “active_until_timestamp_requested” is provided in the data and set to “true” (or 1): An object containing the “active_until_timestamp” (“auts”) which is the time at which the PAYGO credit of the device will run out as a UNIX timestamp. **This is not secure and is meant for debugging.**

```
{"auts": 1616081621}
```

Get Device Data (Optional)

Method: GET

Route: /device_data or /dd

Request Parameters:

- **from_datetime**: Datetime in ISO 8601 format for the beginning of the time for which you want the historical data.
- **to_datetime**: Datetime in ISO 8601 format for the beginning of the time for which you want the historical data.

Description: You can use this route to get a collection of device data from a server.

Expected Content: A Main Object matching the Main Object Data Structure. A server will always return it in the Simple format.

Register Device Data Format

Method: POST

Route: /data_format

Description: You can use this route to send a collection of device data to a server.

Expected Content: A Data Format Object matching the Data Format Data Structure.

Expected Response: A 201 code with an ID of the registered data format object.

```
{"id": 123}
```

Server communication

Base URL

The base url for the API of a platform is provided by the owner of the server and might contain not only the domain and extension but a route (e.g. "https://example.server.com/exampleroute/").

Routes are provided relative to that base URL, for example the route "/device_data" is located at https://example.server.com/exampleroute/device_data in our example.

Security

A. HTTPS with TLS, transport layer security

The connection to the platform must be secured by HTTPS using TLS v1.1 or TLS v1.2 or TLS v1.3. The supported cipher suites are EECDH+AESGCM, EDH+AESGCM, EECDH+AES256, EDH+AES256.

B. JWT Key, request origin and permission verification

The HTTP request header must include a valid JWT key provided in the "Authorization" header and preceded by the keywork "JWT" and a space. The JWT key can be obtained from the owner of the server and will be valid only for that server. For example:

`Authorization:` `JWT xxxxxxx.yyyyyyyyy.zzzzzzz`

Encoding rules (JSON or CBOR)

All content exchanged with the API needs to be in JSON format as defined by RFC-8259. Date and datetimes needs exchanged must be in the format specified by ISO 8601 (compatible with RFC3339). The "content-type" header should be set to "application/json" or simply "json".

Alternatively, the content exchanged may be encoded using CBOR to reduce data usage, in which case the header "content-type" should be set to "application/cbor" or simply "cbor".

Data to be stored for each server

To allow for easy interoperability, a device needs to store only a few characteristics for each of the servers it wants to connect to. The value for those settings can be setup in factory or at the distributor directly depending on the manufacturer's choice. The key values to store are:

- The **base URL** that is used to know where to send the request to
- The **JWT key** that is specific for each of the server and should be obtained from the owner
- The **Data Format ID** that is different for each server. We highly recommend registering the data format in advance in each of the servers and then just storing the IDs for use in the request.