Fleetistics IoT
Data Process Service IoT API

Data Processing Service (IoT) API

The Fleetistics DPS-API provides unified access to the information reported by any device registered in your account. Devices of all manufacturers, type, configuration and device data provider are supported. For all device types, the system provides a list of registered devices.

Information provided includes:

- A list of each device's current state, manufacturer/type/configuration, current location, speed, battery status, last update date.
- A list of raw device parameter values for specific devices by date/time. The list of parameters is unified and independent from device types. However, each device type or configuration can send its own set of parameters.
- A list of raw device messages for specific devices by any date/time period. Messages contain only updated device parameters and location. Messages always provide a unified type reliant on the device message type. If device sends events such as start/stop motion or ignition on/off, the message will contain a unified event type. Messages can be empty if no change in parameter value or status.

Required skills: The ability to form and send an HTTP request with header and handling for JSON responses.

To get started: In MyFleetistics, go to Account > Services > Administer Services. Enable the Data Processing Service (IoT) API option. Then open Service Settings by clicking the icon to obtain an API Key. The Service Settings page contains an API Explorer hat includes a list of all requests and the ability to test each request on your real data, then copy the request URL.

Request details:

1. To get data send an HTTP Get request with parameters that include a query string and data will be returned as JSON response.

   Request must include header "Authorization" with value "Bearer " + APIToken.Z

   You can copy "Authorization" header on from the Data Processing Service (IoT) settings by clicking the
   Copy Auth Header button.

2. Requests can be concurrent.

3. Max number of concurrent requests and max number of requests per hour are limited by the service level selected. If requests exceed the limit - HTTP code 429 Too Many Requests is returned.

4. The maximum number of returned messages by single request (GetDeviceMessagesBy*/GetMessagesBy*) is limited by 1000 messages. If a request returns more than 1000 messages - only first 1000 messages are returned and the flag "isLimited" is true. Returned messages are ordered by Message.Id so you can build new request starting from the last returned Message.Id

5. The base API url is: https://my4.myfleetistics.com/api/dps/
6. A call always returns a JSON object containing the fields "result" and "payload". The response result=0 indicates success. The response result !=0 indicates an error. An error returns the field "message" with details.

7. Payloads can be a JSON object or array depending on the request

Examples of calls for use with GPS devices:

1. Show devices on map or show list of devices statuses:

   - Request the list of devices once (/GetDevices) – to show device name and type. ·
   - Periodically request the list of device statuses (/GetDeviceCurrentStatuses)

   Device current status contains latest device location, speed and bearing to show position on a map. It also includes LastCommunicationDate, IsBatteryOk, IsIgnitionOn, IsInMotion for device state.

2. Show device state for a date/time

   · Request the list of devices (/**GetDevices**) – to show the device selector
   · Ask user for **DateTime**
   · Request a list of parameters (/**GetParameterTypes**) - to display parameter names.
   · Request initial device state (device parameters values) on “**DateTime**” (/**GetDeviceStateUpToDate**)

   Showing device location and all parameter values can be accomplished with **DateTime**

3. Show device activity:

· Request device list (/GetDevices) – to show the device selector.
· Ask user for fromDate and toDate.
· Request list of parameters (/GetParameterTypes)- to show parameter names.
· Request list of event types (/GetEventTypes)- to show eventTypes names.
· Request initial device state (device parameters values) using “fromDate” (/GetDeviceStateUpToDate)

All device parameters values at the beginning by entering user period.

- To request device messages for period fromDate to toDate (/GetDeviceMessagesByDate). If the quantity of messages is more than 1000, the response will include **isLimited** = true. In this case – send again (/**GetDeviceMessagesByDate**)with the same period **fromDate** to **toDate** adding parameter **fromMessageId** as **maxMessageId**+1 from latest response to build afull list of messages.

Device messages contain only device parameters location changed by this message. Also some devices send events like start/stop motion or ignition on/off – in this case the message will contain the event. Some messages can be empty if nothing changed. Empty messages should be ignored and show only non-empty messages. This allows the user to see device activity during the period.

4. Show device messages

· Request list of devices (/GetDevices) – to show the device selector.
· Ask user for fromDate and toDate.
· Request the list of parameters (/GetParameterTypes)- to show name of parameters.
· Request the list of message types (/GetMessageTypes) - to show name of messageTypes
· Request the list of event types (/GetEventTypes)- to show name of eventTypes
· Request device messages for period fromDate to toDate (/GetDeviceMessagesByDate).

If the quantity of messages is more than 1000, the response will include **isLimited** = true. In this case – send (/**GetDeviceMessagesByDate**) again with the same period **fromDate** to **toDate** including the additional parameter **fromMessageId** as **maxMessageId**+1 from the latest response to return the full list of messages.

When the list of messages is called, the message type can be a periodical update message or system message. If message has event – the event name is included. Empty messages can be shown or ignored. Some empty messages can be system or contain only an event.

If full device state for each message is needed, load (/GetDeviceStateUpToDate) at the beginning of period, then check that messages are sorted by messageId in the list. In series replace the values in full device state with the same parameters. Location is provided in device state for each message.

5. Datafeed service for continuously syncing all data to an internal database.

· Request the list of devices once, parameters, message types, event types (/GetDevices, /GetParameterTypes, /GetMessageTypes, /GetEventTypes) to save them into your database.
· Periodically, for example, once a day, request the lists again to check using “LatestUpdateDate” to determine is an entity changed, updadted, or add it to your database.
· Request the initial device state (device parameters values)once for the start date of a datafeed (/GetDevicesStateUpToDate)request to save to a database. This request return states of all devices. The datafeed process can be restarted anytime from the same or a different date
· Request all device messages once for all devices for a DataFeed start date without specifying a to date (/GetMessagesByDate). If the quantity of messages is more than 1000, the response will include **isLimited** = true. In this case send (/**GetMessagesByDate**)again with the same **fromDate** but include parameters **fromMessageId** as **maxMessageId**+1 from latest response. This builds the full list of messages. When all messages are returned, save the latest **maxMessageId** to a database.
· To periodically calculate period how frequently devices are reporting, how frequently data is to be updated, and what are the requests limitations on your account, request new messages using (/GetMessagesById) with latest messageId saved to a database.If the quantity of messages is more than 1000 use the same procedure as the initial load but with (/GetMessagesById). Once all messages are received, save the latest messageId to a database.

To implement a datafeed of device messages, request a list of available devices again.

Then, periodically sync the list of devices with returns with usage device.Id as unique ID.

Device messages can be requested per device or for all devices. Message.Id can be used as serial.

It is integer incrementing messages unique key that is never changed.

Message.DCSDate is UTC time when server received message from device.

It never returns lower number than those already received.

Message.Date is generated by device. If a device sends delayed messages from its cache, you can receive an older date than those already delivered.

6. Base API url is: “https://my4.myfleetistics.com/dcsapi/”

7. A JSON object with fields "result" and "payload" are always returned. Result=0 indicates success, result !=0 indicates an error. In case off error a "message" with details is returned. Payload can be a JSON object or array depending on the request.

8. Device messages contains two dates:

   - Date - is the DateTime of message sent by device.

   - ServerDate - is the date the server received the message.

   - Both dates are in UTC. Device messages can be stored in device memory and sent at a later time.

DeviceCurrentStatus contains deviceID - to show device name or type, use the result of /GetDevices request.

GetDeviceCurrentStatuses does not return current status for a device that has never reported.

GetDeviceCurrentStatuses returns one current status per device.

GetDeviceCurrentStatuses always returns a list of all current statuses. It does not depend on reported device since your previous

GetDeviceCurrentStatuses request or no.

API Requests:

1. To return a list of Parameter, Message, or Event types use (GetParameterTypes, GetMessageTypes, GetEventTypes). Example:
https://my4.myfleetistics.com/dcsapi/GetParameterTypes returns:
"Id" - integer, ID of Parameter, Message, Event, used in the rest of requests
"Name" - string, name of Parameter, Message, Event
"LatestUpdateDate" - integer, Unix time, UTC/GMT, time of latest entity parameter update
Parameters, Message types, Event types include device manufacturer, type, configuration independence.

Sample:

{

"result":0,

"payload":{ “messageTypes”: [

{

"Id":4,

"Name":"Update",

"LatestUpdateDate":1564156521

},{

"Id":17,

"Name":"System",

"LatestUpdateDate":1566797645

}

]

}

}

2. List of devices, /GetDevices returns:

"Id" - integer, ID of device will be used in the rest of requests
"TypeName" - string, name of device type
"SerialNumber" - string, unique ID of device, depending on the device type, this can be the device IMEI, ESN, UID, Serial number.
"Name" - string, name of device in MyFleetistics
"ActivatedDate", integer, Unix time, (number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT)), UTC/GMT. The time the device started reporting for into this account
"DeactivatedDate" - integer, Unix time, UTC/GMT, time when the device stopped reporting for into this account, optional, can be null for active device
"LatestUpdateDate" - integer, Unix time, UTC/GMT, time of latest device parameter update
"LatestDeviceMessageDate" - integer, Unix time, UTC/GMT, time of latest device message, optional, is null for non-reporting device

Example:

{

"result":0,

"payload":{

“devices”: [

{

"Id":4,

"TypeName":"FLT-TLP",

"SerialNumber":"866425035335092",

"Name":"TopFly 0x26 Test",

"ActivatedDate":504921600,

"DeactivatedDate":1429825293,

"LatestUpdateDate":1564156521,

"LatestDeviceMessageDate ":1564156521

},{

"Id":17,

"TypeName":"FLT-TLP",

"SerialNumber":"866425039637030",

"Name":"test DCS export",

"ActivatedDate":1566297674,

"LatestUpdateDate":1566797645,

"LatestDeviceMessageDate ":1564156521

}

]

}

}

3. List of devices current statuses, /GetDevicesCurrentStatus this request returns statuses of reporting devices Device type independent response. LastCommunicationDate, IsBatteryOk, IsIgnitionOn, IsInMotion **"LastCommunicationDate"** - integer, Unix time, UTC/GMT, time of latest device message of any type - position message, alert, heartbeat or system message **"IsBatteryOk"** - boolean, multi-sourced battery OK status, obtained from different parameters returned by different types of devices. Includes internal/external/backup battery state. **"IsIgnitionOn"** - boolean, current state of ignition if device supports it. Absent if device does not send ignition state **"IgnitionChangeDate"** - integer, Unix time, UTC/GMT, time when ignition state last changed **"IsInMotion"** - boolean, multi-sourced status obtained from non-zero speed and in motion state returned by some devices **"InMotionChangeDate"** - integer, Unix time, UTC/GMT, time when motion state last changed **"PositionDate"** - integer, Unix time, UTC/GMT, time when latest valid position was received from device **"Latitude"**, **"Longitude"** - double, latest valid GPS position **"Bearing"** - float, in degrees, latest vehicle direction received from device **"Speed"** - float, in km/h, latest device speed received from device. If the device sends both OBD and GPS speed - OBD speed will be used Sample: { "result":0, "payload": { “devicesCurrentStatus“: [{ "DeviceId":4, "LastCommunicationDate":1564156487, "IsBatteryOk":true, "IsIgnitionOn":true, "IgnitionChangeDate":1564145612, "IsInMotion":false, "InMotionChangeDate":1563403548, "PositionDate":1564145926, "Latitude":28.187429428100586, "Longitude":-82.374893188476562, "Bearing":291, "Speed":0.0 },{ "DeviceId":17, "LastCommunicationDate":1566320213, "IsIgnitionOn":false, "IgnitionChangeDate":1566320213, "IsInMotion":false, "InMotionChangeDate":1566320213, "PositionDate":1566320213, "Latitude":28.19282, "Longitude":-82.37767, "Bearing":143, "Speed":0.0 }] } }

4. Device messages Can be used for a single device - **/GetDeviceMessagesByDate, /GetDeviceMessagesByServerDate, /GetDeviceMessagesById** with required parameter- **deviceId** or for all devices - **/GetMessagesByDate, /GetMessagesByServerDate, /GetMessagesById** Device message contain two dates: - Date - is the DateTime of message sent by device - ServerDate date time - is the date the server received the message. Both dates are UTC. A message can be stored in a devices memory and sent later when the device has a network connection. So message.ServerDate can be later than message.Date **/GetDeviceMessagesByDate, /GetMessagesByDate** have optional parameter - **fromDate**, **toDate -** integer, Unix time, UTC/GMT, message Date **/GetDeviceMessagesByServerDate, /GetMessagesByServerDate** have optional parameter - **fromServerDate**, **toServerDate -** integer, Unix time, UTC/GMT, message ServerDate **/GetDeviceMessagesByDate, /GetDeviceMessagesByServerDate, /GetMessagesByDate, /GetMessagesByServerDate** have optional parameter – **fromMessageId** – integer, min messageId to get next pack of messages if a previous request response was limited due to quantity of messages **/GetDeviceMessagesById, /GetMessagesById** have optional parameter - **fromId**, **toId -** integer, message Id The set of returned parameters depends on device type/configuration. Parameter value, type and formatting are device independent. **"Id"** - integer, unique ID of message. **"DeviceId"** - integer , unique ID of device **"Date"** - integer, Unix time,

UTC/GMT, device generated date/time of message **"ServerDate"** - integer, Unix time, UTC/GMT, date/time when server received message **"MessageTypeId"**, integer, unique ID of message type. Can be decoded to name of message type by the list returned by /**GetMessageTypes "EventTypeId"**, integer, unique ID of event type. Can be decoded to name of event type by the list returned by /**GetEventTypes** An optional parameter is sent when a device sends an event like ignition on, vibration, accident, OBD errors: **"Parameters"** – returns a list of parameters changed by message. Can be omitted if message does not change any parameters Parameter structure: **"TypeId"**, integer, unique ID of parameter type. Can be decoded to name of parameter type and unit by the list returned by /**GetParameterTypes "Value"** - string representation of value **"Position"** - GPS position if position changed. Can be omitted if message does not change the position Position structure: - "Latitude", "Longitude" - double, GPS position - "Altitude" - float, altitude, m **"Quality"** - integer, can be GPSFine = 1, GPSFair = 2, GPSPoor = 3, LBS = 4, Interpolated = 5, GPSBad = 13 Sample: { "result":0, "payload":{ messages: [{ "Id": 12, "DeviceId":11, "Date":1561100731, "ServerDate":1561115135, "MessageTypeId":1, "EventTypeId": 1 , "Parameters":[{ "TypeId": 1, "Value":93.0 },{ "TypeId":2, "Value":0 },{ "TypeId":3, "Value":4153344.0 },{ "TypeId":4, "Value":1 }], "Position":{ "Latitude":29.026450994783758, "Longitude":-98.360950751417988, "Altitude":0.0, "Quality":0 } }], maxMessageId: 12, isLimited: false } }

5. Device state to a date Can be requested for single device - /**GetDeviceStateUpToDate** with required parameter- **deviceId** or for all devices using /**GetDevicesStateUpToDate** both requests have required parameter - **upToDate** - integer, Unix time, UTC/GMT – message Date. Device state is built for latest message Date < **upToDate** Single device request has payload with single status object and all devices requested will have array of status objects. Status object will have: **“DeviceId”** – integer – Id of Device **"MaxMessageId"** - integer, max messageId with Date < **upToDate "Parameters"** – this section has the same structure as message parameters but contains all possible device parameters **"Position"** - has the same structure as message Position. Parameters and Position sections can be omitted if device did not send position or parameters up to **upToDate** Samples: **For single device** { "result":0, "payload":{ “state”: { "DeviceId": 23, "MaxMessageId": 23, "Parameters":[{ "TypeId": 1, "Value":93.0 },{ "TypeId":2, "Value":0 },{ "TypeId":3, "Value":4153344.0 },{ "TypeId":4, "Value":1 }], "Position":{ "Latitude":29.026450994783758, "Longitude":-98.360950751417988, "Altitude":0.0, "Quality":0 } } } } **For all devices** { "result":0, "payload": { “states”: [{ "DeviceId": 23, "MaxMessageId": 23, "Parameters":[{ "TypeId": 1, "Value": 93.0 },{ "TypeId": 2, "Value": 0 },{ "TypeId":3, "Value": 4153344.0 },{ "TypeId":4, "Value":1 }], "Position":{ "Latitude":29.026450994783758, "Longitude":-98.360950751417988, "Altitude":0.0, "Quality":0 } }, { "DeviceId": 24, "MaxMessageId": 26, "Parameters":[{ "TypeId": 1, "Value": 93.0 },{ "TypeId": 2, "Value": 0 },{ "TypeId":3, "Value": 4153344.0 },{ "TypeId":4, "Value":1 }], "Position":{ "Latitude":29.026450994783758, "Longitude":-98.360950751417988, "Altitude":0.0, "Quality":0 } }] } }

6. Refresh token API token expires after 1 year. Before a token expires – new tokens can be generated for next 1 year period by request /**RefreshToken** Old

token will not valid Sample { "result":0, "payload": { “newToken”: “zzxxcc…” } }

Unique solution ID: #1021
Author: n/a
Last update: 2021-06-03 16:24