Data Intake Gateway (DIG)

API Endpoint Workflows for Records Ingestion and Invalid Records

**Table of Contents**

[**Introduction**](#_hoqkvvr9mu8b) **3**

[Purpose](#_z7fhlly5rpbg) 3

[Terms and Definitions](#_739pq19sq7el) 3

[Required Reading](#_w4qpwjx1es8s) 3

[Overview](#_jp36pi9ztf86) 4

[**Records Workflow**](#_opewb6fg3yvi) **4**

[Records Endpoint](#_l8xuvf6bt4l3) 5

[Data Integrity Considerations](#_b18vhd4j8z39) 7

[Size and Frequency of Bulk Requests](#_jl12qhfx0op7) 7

[Curl Example](#_zfq17whfht3h) 8

[Invalid Records Workflow](#_yocbkc9nnj5v) **8**

[Explanation for NextResultKey and CurrentResultKey](#_9p1c6jvhf15) 11

[Explanation for CurrentNumberOfInvalidRecords and TotalNumberOfInvalidRecords](#_r4qlytdwyj4j) 12

[Curl Example](#_yxnn8w33y9er) 13

[Invalid Record Expiration](#_bstng46pizow) 14

[Workflow Walkthrough](#_a4c41itynbf) 14

[**Best Practices & Usage Expectations**](#_p19mc7m7gezh) **18**

[/authentication/\*](#_3r0tnid8q85e) 18

[Separation of the /records Pipeline & /invalidrecords Processes](#_q8eu7prdpzuc) 18

[**Appendix**](#_d89yrubytljl) **19**

[Custom Telematics Device Records](#_cvx7jvdttg20) 19

[GpsRecord](#_xge607gwapfb) 19

[StatusRecord and GenericStatusRecord](#_ezqulq2bpgvp) 20

[BinaryRecord: Storage of Arbitrary Binary/String Data](#_8323p6b5nskz) 20

[DriverChangeRecord: Driver Assignment](#_2p0vbam7945f) 21

[Fault Records](#_9n67m0e9g90g) 22

# Introduction

## Purpose

This document outlines the non-authentication API records-related endpoint workflows for the Data Intake Gateway (DIG). These include:

* Posting records into DIG — **/records POST**
* Querying for invalid records — **/invalidrecords GET**

The document also includes best practices and sample implementation flow charts for each of the workflows.

| ✱ **NOTE**: See [Data Intake Gateway (DIG) Guide](https://docs.google.com/document/d/15uNuPqwFcPLe6vKs_JgY5nPTy2isQ3WYUu4oyQ3cEfQ/edit#heading=h.pg9nc6swxzvc) for more information on the solution. |
| --- |

## Terms and Definitions

| **Term** | **Definition** |
| --- | --- |
| Record(s) | Contains context-specific information for a specific device. Usually associated with a specific point in time as a snapshot of data. All records for DIG are considered Custom Telematics Device Records and these terms are used interchangeably. |
| Custom Telematics Device Record(s) | Data records from custom telematics devices. There are numerous types of records and so the content within each record is contextual to the record type itself. |
| Accepted Record(s) | A Geotab record that conforms to the record specification (data model) and is accepted in the **/record** DIG endpoint. |
| Invalid Records(s) | Records that have correct specifications and data model conformity but are deemed invalid subsequently for a specific reason. These are available for inspection via the **/invalidrecords** DIG endpoint. |
| Valid Records(s) | A Geotab record that meets all required criteria to be processed in MyGeotab. |
| DateTime Timestamp | All DateTime fields for DIG API endpoints must conform to [ISO 8601](https://www.iso.org/iso-8601-date-and-time-format.html) [RFC3339](https://tools.ietf.org/html/rfc3339) timestamp format in UTC or offset timezone (e.g. 2020-01-01T01:00:00Z). |

## Required Reading

Before reading this document, you should read and understand:

1. The [DIG API Authentication Workflow](https://docs.google.com/document/d/1aRPIDz7d49BEqEID_ZLjtrhwAuHacWO1WhTMUVUXwMI/edit#). All workflows identified here require a valid and unexpired bearer token. All DIG API endpoints (that are not authentication related endpoints) require the **Authorization: Bearer** HTTP header. Thus, integration systems need to implement the authentication workflow alongside the aforementioned flows.
2. The above Terms and Definitions.

## Overview

The DIG API provides a primary endpoint to enable integrators to send bulk custom telematics device records to MyGeotab using the **/records** endpoint.

Additionally, it provides secondary endpoints to help the integrators with troubleshooting, which include:

* The **/invalidrecords** endpoint for invalid records that may have been deemed as invalid or rejected.

# Records Workflow

Systems that integrate with DIG are expected to send bulk requests containing valid records to the **/records** API endpoint. You can make these requests as frequently as required for your use case with a consideration of size per request. See the best practices for [/records](#_q8eu7prdpzuc) to learn more.

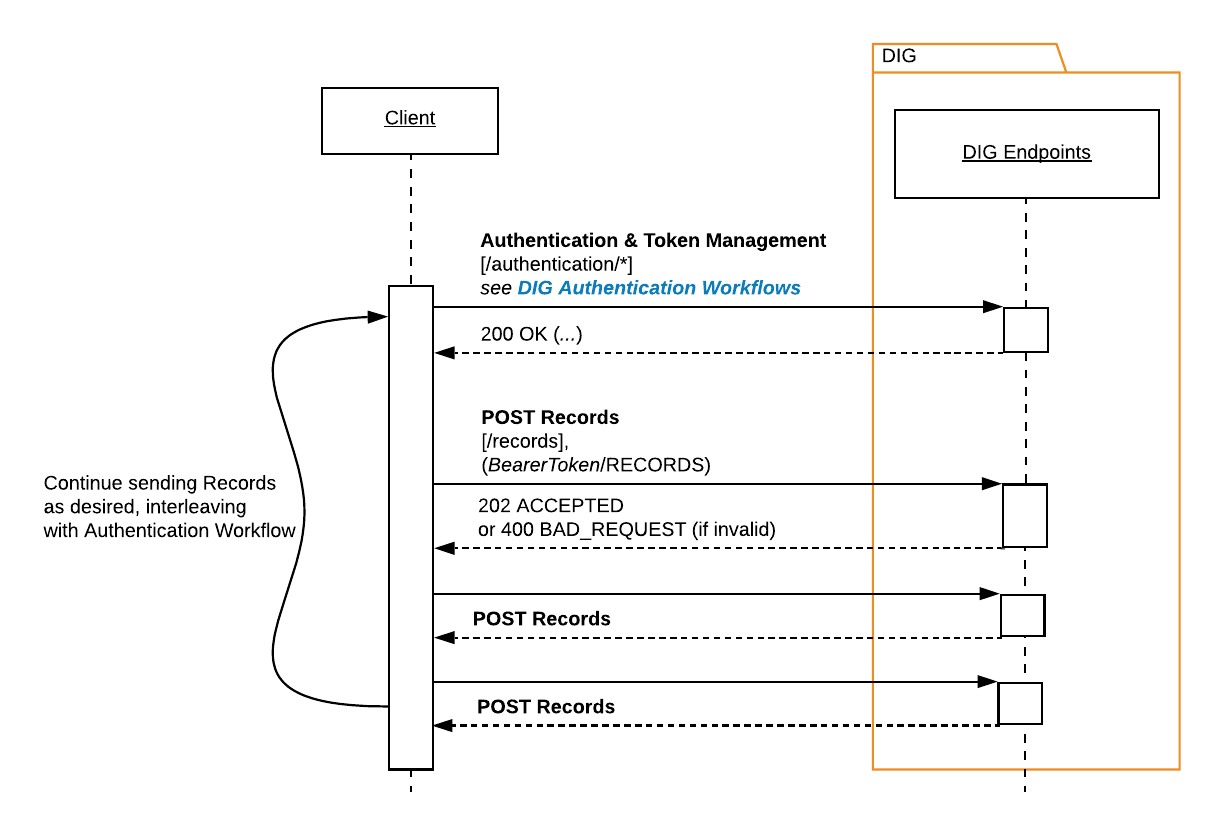
[](https://lucid.app/documents/edit/4e52f742-5be8-47e9-a31d-86eea3cfcbb0/1?callback=close&name=docs&callback_type=back&v=2915&s=612)

Figure 1 - Record Endpoint API Workflow

## Records Endpoint

A call to **/records** posts a data payload of records using JSON format as an array of record items. All records must contain the following [string] fields:

* “DateTime” (timestamp)
* “SerialNo” (device serial)
* “Type” (record type)

You should provide all other record fields necessary for the given record type. See the [DIG Open API YAML spec](https://github.com/Geotab/data-intake-gateway/blob/develop/data-intake-gateway-api.yaml) for a list of supported record types and their respective fields.

| ✱ **NOTE:** Some record types may require additional context and understanding. Please see [Custom Telematics Device Records](#_cvx7jvdttg20) in the appendix below or reach out to [integrations@geotab.com](mailto:integrations@geotab.com) for support. |
| --- |

| ✱ **NOTE:** Data accepted by DIG is not immediately available in MyGeotab. Backend solution processing takes place, and the data becomes available in the target database(s) as quickly as possible. |
| --- |

| **Term** | **Definition** |
| --- | --- |
| Accepted Request | A valid request to /records that conforms to the HTTP POST specification requirements, contains properly formatted JSON, and contains acceptable records. It returns an HTTP 202 ACCEPTED response. |
| Invalid Request | A request to /records that is rejected because it is not valid. The entire request is rejected with HTTP 400 BAD\_REQUEST response. |

See a sample workflow implementation flowchart below:

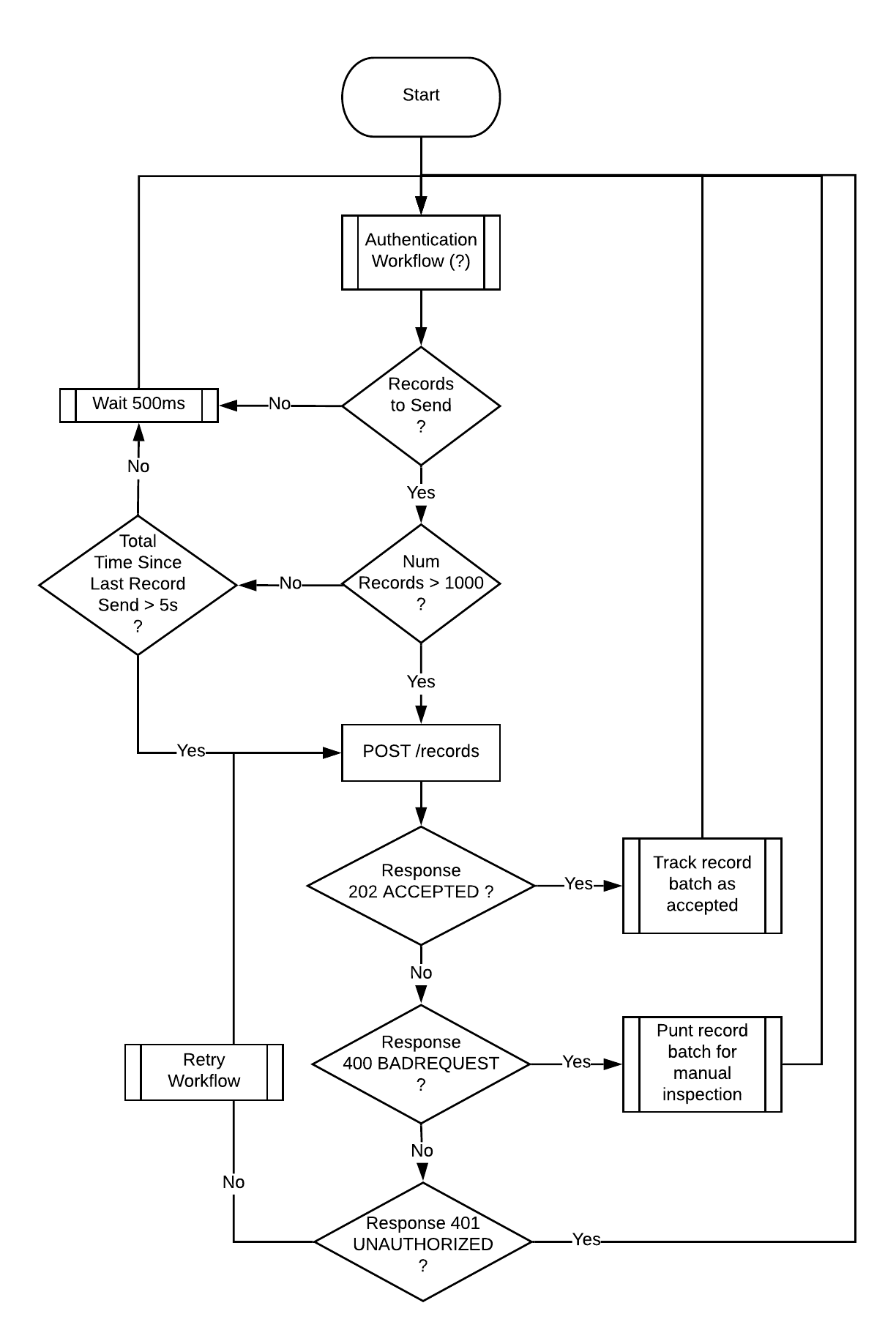
[](https://lucid.app/documents/edit/4e52f742-5be8-47e9-a31d-86eea3cfcbb0/3?callback=close&name=docs&callback_type=back&v=2915&s=612)

Figure 1 - Record Endpoint Sample Implementation Workflow

**Endpoint**

[https://dig.geotab.com:443/records] **POST**

**Request Body**

Array of valid records items

**Returns**

If the request is accepted:

* **202 ACCEPTED** with
  + Data: for successful responses, returns a request ID.
  + Error: for unsuccessful responses, populated with error message(s).

If the request is NOT accepted:

* **400 BAD\_REQUEST** with error message(s); or
* **401 UNAUTHORIZED** with error message(s).

The above responses should not be automatically retried, as they require corrective action.

Other HTTP response codes may be retried per their specifications, such as:

* **503 SERVICE\_UNAVAILABLE** with error message(s)

## Data Integrity Considerations

Please consider the following:

| ✱ **NOTE:** The solution ignores extraneous data fields in records that do not conform to the record specification. |
| --- |

| ✱ **NOTE:** Optional fields with null values or that are missing entirely are set to a default value following the [Default values of C# types](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/default-values). |
| --- |

## Size and Frequency of Bulk Requests

The frequency and size of requests can have competing priorities. Consider that typically:

* Smaller bulk requests have lower throughput performance with lower latency.
* Larger bulk requests have higher throughput performance with increased latency towards the last record in the batch.

There is no "one size fits all". Integrators need to determine what works best for their systems. The example flowchart above uses sane default starting values.

| ✱ **NOTE:** If you need to send larger than normal bulk processing requests, we advise that you do it during off-peak hours (relative to the timezone of the region). Please contact [Geotab Support](https://myadmin.geotab.com/helpdesk) before sending extremely large requests that are out of the ordinary; 2-3 business days advanced notice may be necessary. |
| --- |

## Curl Example

**Request**

curl --request POST \

--url https://dig.geotab.com:443/records \

--header 'Authorization: Bearer eyJh**--Bearer-token-clipped-for-brevity-**TCsIA' \

--header 'content-type: application/json' \

--data '[{RECORD\_1}, {RECORD\_2}, {...}, ...]'

Where: RECORD\_X (in example) could be a GenericStatusRecord:

{"DateTime":"2020-01-01T01:00:00Z","SerialNo":"xxxxxxx","Type":"GenericStatusRecord","Code":X,"Value":X}

Response:

{

"Error": [],

"Data": "6184f7b2-**clipped-for-brevity**-fa93c1e53a5d"

}

# Invalid Records Workflow

DIG provides the **/invalidrecords** endpoint to query for invalid records that may have been marked as invalid or rejected for various reasons. Systems that integrate with DIG should periodically query this endpoint to inspect, identify, and correct any problems. Records that are corrected can then be resent to the DIG **/records** endpoint.

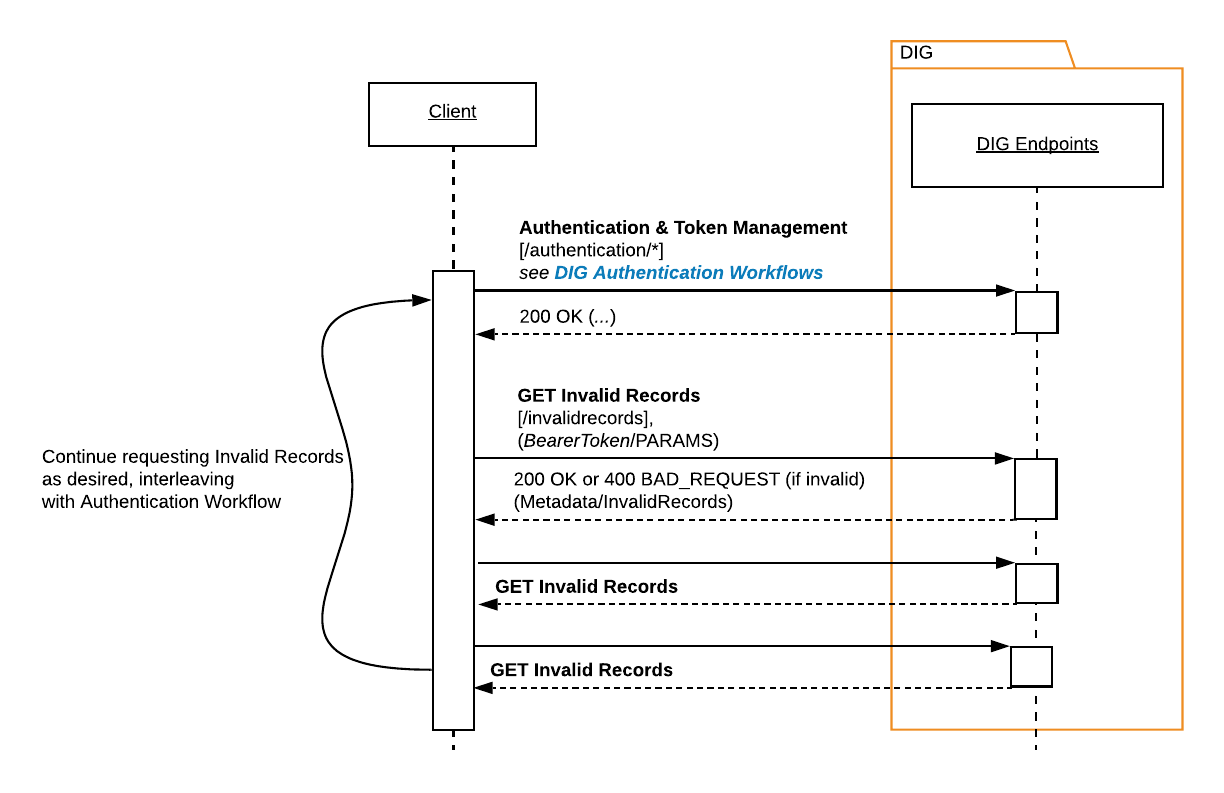
[](https://lucid.app/documents/edit/4e52f742-5be8-47e9-a31d-86eea3cfcbb0/2?callback=close&name=docs&callback_type=back&v=2915&s=612)

Figure 3 - Invalid Record API Endpoint Workflow

Invalid records accumulate within DIG as they are processed and identified through the [**/records** workflow](#_opewb6fg3yvi). Invalid records are not sent to MyGeotab; they become available for your account to query and inspect as soon as possible.

A request to **/invalidrecords** gets a data payload that may contain invalid records if they exist. The workflow and usage of the endpoint supports some basic parameters for controlling the maximum number of results per response and the next result key for walking results (aka pagination) and incremental calls over time. The response also includes metadata that is useful for the integrator to implement this workflow with the system.

The array of returned invalid records is a consistent structure that includes their type, original field values, the reason why they are considered invalid, along with other metadata.

DIG will expire invalid records after a period of time. To avoid data loss, it is expected that your integration queries this endpoint periodically to save them for manual inspection.

To recap: The integrator must periodically query **/invalidrecords** to investigate erroneous data, correct problems, and resend the data.

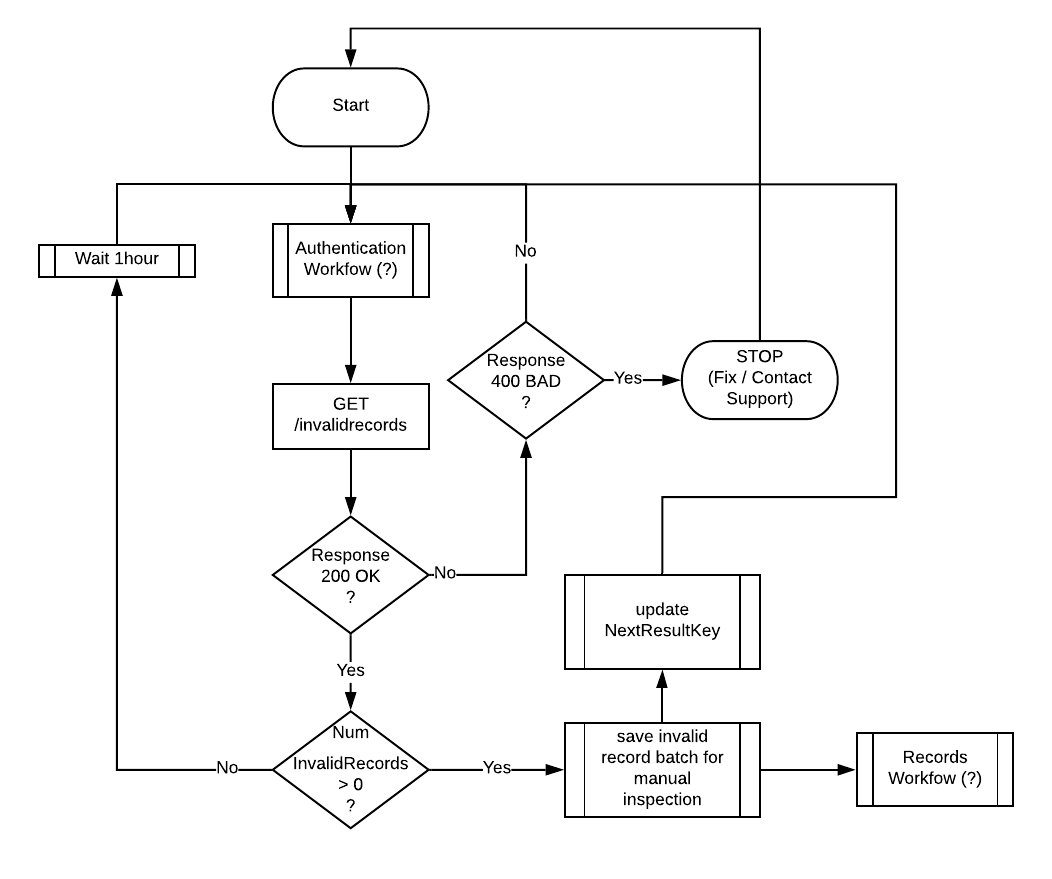
[](https://lucid.app/documents/edit/4e52f742-5be8-47e9-a31d-86eea3cfcbb0/4?callback=close&name=docs&callback_type=back&v=2915&s=526)

Figure 4 - Invalid Records Endpoint Sample Implementation Workflow

**Endpoint**

[https://dig.geotab.com:443/invalidrecords] **GET**

**Params**

NextResultKey: set to 0 initially; subsequent requests should set this to the previous request’s NextResultKey response value;.

NumberOfResults: the maximum number of invalid records to return for this request

**Returns**

If the request is accepted:

* **200 OK** with
  + Data (for successful responses):
    - **NextResultKey**: use this for the next request’s NextResultKey parameter.
    - **CurrentResultKey**: the value of NextResultKey that was specified in the current request.
    - **TotalNumberOfInvalidRecords**: the total number of unexpired invalid records for your account.
    - **CurrentNumberOfInvalidRecords**: the number of invalid records returned in this response.
    - **InvalidRecords**: [array] of invalid records (if any) with metadata.
  + Error (for unsuccessful responses, populated with error message)

If the request is NOT accepted:

* **400 BAD\_REQUEST** with error message(s); or
* **401 UNAUTHORIZED** with error message(s).

See the [DIG Open API YAML spec](https://github.com/Geotab/data-intake-gateway/blob/develop/data-intake-gateway-api.yaml) to learn more about the response metadata and InvalidRecord metadata.

## Explanation for NextResultKey and CurrentResultKey

There is a relationship between:

1. **NextResultKey** in a request and **CurrentResultKey** in the response.
2. **NextResultKey** from a response and **NextResultKey** in the subsequent request.

**NextResultKey** can be thought of as a pagination key that will unpredictably increase monotonically, and must be tracked and maintained by the integration, as it is chained from request to request. If this tracker is lost, the integration will need to start over with 0 and would be forced to reprocess duplicate invalid records. Note that this value is not guaranteed to always increase.

**CurrentResultKey** in a response is simply metadata equal to the value of **NextResultKey** from the associated request.

| ✱ **NOTE:** The NextResultKey value in the response is not predictable. Your integration implementation must take care to adhere precisely to this value in subsequent requests. The value in a response may be the same as CurrentResultKey or some numerical value higher than the current result key. The primary function of this value is for the subsequent request’s NextResultKey parameter. |
| --- |

The following simplified sequence diagram can help you understand these relationships:

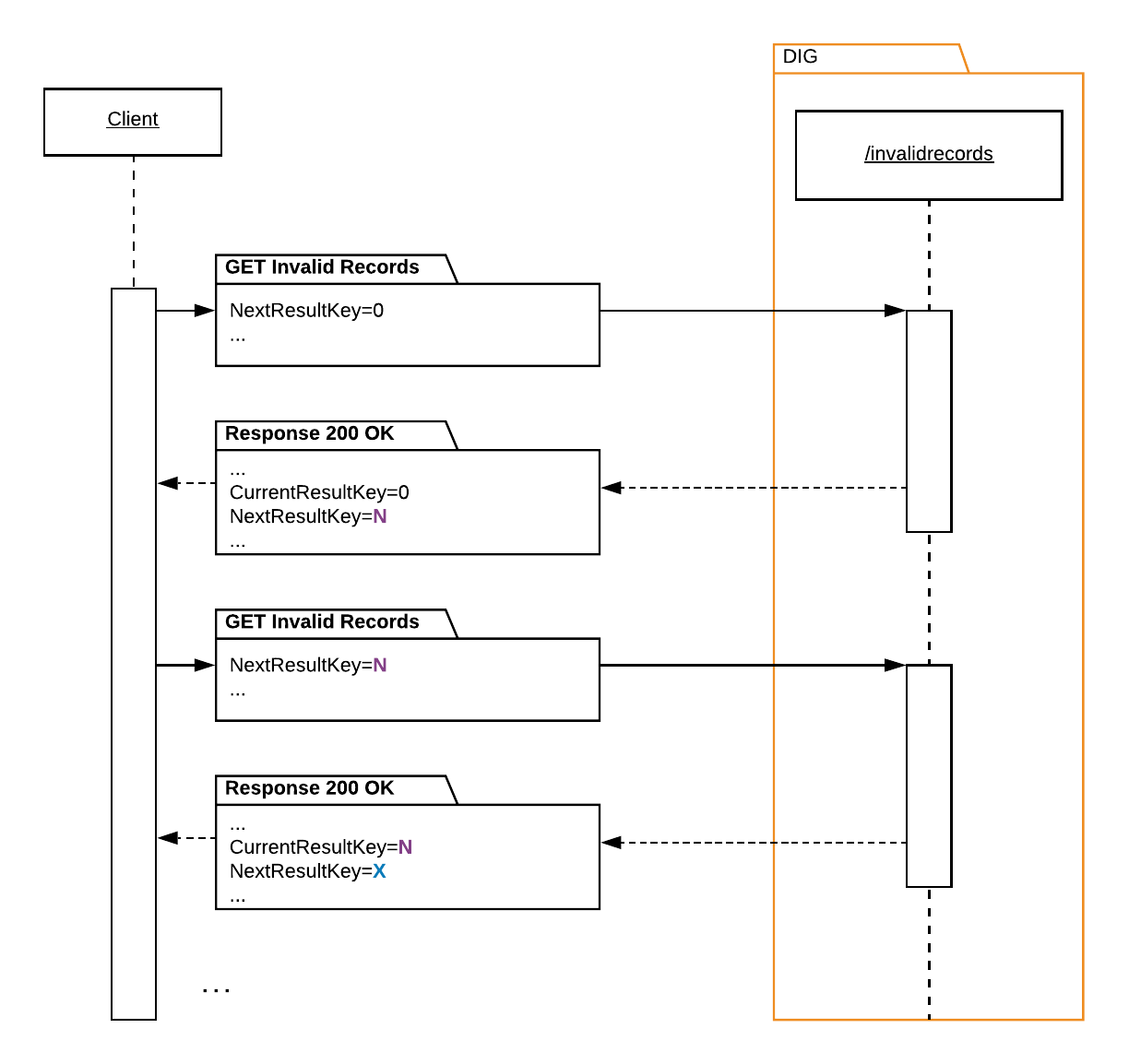
[](https://lucid.app/documents/edit/4e52f742-5be8-47e9-a31d-86eea3cfcbb0/5?callback=close&name=docs&callback_type=back&v=2915&s=612)

Figure 5 - Invalid Records ResultKey Explanation

## Explanation for CurrentNumberOfInvalidRecords and TotalNumberOfInvalidRecords

Both of these response fields are simple counters for the number of invalid records respectively.

**CurrentNumberOfInvalidRecords** will always be a subset of the **TotalNumberOfInvalidRecords**, and will be less than or equal to the request parameter **NumberOfResults**. While **CurrentNumberOfInvalidRecords** is >=0, there may be more invalid records returned in subsequent requests to **/invalidrecords**.

**TotalNumberOfInvalidRecords** will increase or decrease over time, but always reflects DIG’s state of total invalid records for your account at the time the request was made. It increases when there are new invalid records processed and it decreases when invalid records expire.

| ✱ NOTE: When the CurrentNumberOfInvalidRecords response value equals zero, we can assert that there are no more invalid records at this time. The integration can temporarily hold querying for more invalid records until the next query cycle time. |
| --- |

## Curl Example

**Request**

curl --request GET \

--url 'https://dig.geotab.com:443/invalidrecords?NextResultKey=0&NumberOfResults=100' \

--header 'Authorization: Bearer eyJh--Bearer-token-clipped-for-brevity-TCsIA'

**Response**

{

"Error": [],

"Data": {

"NextResultKey": 10514,

"CurrentResultKey": 0,

"TotalNumberOfInvalidRecords": 200,

"CurrentNumberOfInvalidRecords": 100,

"InvalidRecords": [{INVALID\_RECORD\_1}, {INVALID\_RECORD\_2}, ..., {INVALID\_RECORD\_100}]

}

}

Where: INVALID\_RECORD\_X (in example) is an invalid record denoting metadata as well as the original record content ingressed to DIG **/records** endpoint. The metadata will include a description for why this record was not valid:

{"TimeStamp":"2020-01-01T01:00:00Z","Cause":"some reason","UserId":"<>", “BaseRecord”: {...}}

Example of a single invalid record:

{

"Error": [],

"Data": {

...,

"InvalidRecords": [

{

"BaseRecord": {

"Code": 255,

"Value": 0,

"DateTime": "2021-01-01T05:00:00Z",

"SerialNo": "dummy",

"Type": "GenericStatusRecord"

},

"Cause": "The record was not accepted because the serial number is unknown or has a formatting issue. This may be an unprovisioned device or we are experiencing a delay to provision.",

"TimeStamp": "2021-07-16T15:00:14.3370137Z",

"UserId": "<snip>"

}

]

}

}

## Invalid Record Expiration

The set of invalid records for your account becomes unavailable once they expire. Typically, invalid records should be available for approximately 48 hours from the time they were identified and processed as invalid records. This expiration is tracked per invalid record.

Requests to the **/invalidrecords** endpoint will never prematurely expire the retrieved records.

| ✱ **NOTE:** To avoid loss of data, we recommend that you query for invalid records every hour, 24/7. Geotab does not provide a hard time guarantee, and invalid records may expire within the system sooner or later than expected. |
| --- |

## Workflow Walkthrough

The following section displays a more in-depth example of several /invalidrecord requests, demonstrating what we have learned.

At a high level, we will work through the following:

1. As an initial state, no invalid records.
2. Then there are 8 invalid records; query for 5 (get 5).
3. No new invalid records, but still more to get; query for 5 (get 3).
4. We have queried for all the invalid records and there are no new ones; query again for 5 (get 0).
5. Workflow takes a break querying for 1 hour.
6. Resume querying, still no new invalid records.
7. Workflow continues querying for several days, still no new invalid records.
8. Invalid records expire.

| 1 | **Initial State: No Invalid Records.**  **Request**  curl --request GET \  --url 'https://dig.geotab.com:443/invalidrecords?NextResultKey=0&NumberOfResults=5' \  --header 'Authorization: Bearer eyJh--Bearer-token-clipped-TCsIA'  **Response**  {  "Error": [],  "Data": {  "NextResultKey": 0,  "CurrentResultKey": 0,  "TotalNumberOfInvalidRecords": 0,  "CurrentNumberOfInvalidRecords": 0,  "InvalidRecords": []  }  }  Comments:   * No invalid records were returned, so CurrentNumberOfInvalidRecords=0 * No invalid records exist yet for this account, so TotalNumberOfInvalidRecords=0 * Notice an unpredictable NextResultKey was returned, to be used in the next request |
| --- | --- |
| 2 | **There are 8 new Invalid Records from the /records pipeline of your integration.**  **Request**  curl --request GET \  --url 'https://dig.geotab.com:443/invalidrecords?NextResultKey=0&NumberOfResults=5' \  --header 'Authorization: Bearer eyJh--Bearer-token-clipped-TCsIA'  **Response**  {  "Error": [],  "Data": {  "NextResultKey": 51886,  "CurrentResultKey": 0,  "TotalNumberOfInvalidRecords": 8,  "CurrentNumberOfInvalidRecords": 5,  "InvalidRecords": [  {  "BaseRecord": {  "Code": 255,  "Value": 0,  "DateTime": "2021-01-01T05:00:00Z",  "SerialNo": "dummy",  "Type": "GenericStatusRecord"  },  "Cause": "The record was not accepted because the serial number is unknown or has a formatting issue. This may be an unprovisioned device or we are experiencing a delay to provision.",  "TimeStamp": "2021-07-16T15:00:14.3370137Z",  "UserId": "<snip>"  },  {...},  {...},  {...},  {...}  ]  }  }  Comments:   * 5 invalid records were returned, so CurrentNumberOfInvalidRecords=5, and the “InvalidRecords” array is populated with the record that was invalid as well as some metadata * 8 invalid records exist for this account, so TotalNumberOfInvalidRecords=8 * Notice again an unpredictable NextResultKey was returned, to be used in the next request |
| 3 | **There are no new Invalid Records from the /records pipeline of your integration. However, some still remain to be queried.**  **Request**  curl --request GET \  --url 'https://dig.geotab.com:443/invalidrecords?NextResultKey=51886&NumberOfResults=5' \  --header 'Authorization: Bearer eyJh--Bearer-token-clipped-TCsIA'  **Response**  {  "Error": [],  "Data": {  "NextResultKey": 51889,  "CurrentResultKey": 51886,  "TotalNumberOfInvalidRecords": 8,  "CurrentNumberOfInvalidRecords": 3,  "InvalidRecords": [  {...},  {...},  {...}  ]  }  }  Comments:   * 3 invalid records were returned, so CurrentNumberOfInvalidRecords=3, and the “InvalidRecords” array is populated * Still 8 invalid records exist for this account, so TotalNumberOfInvalidRecords=8 * Notice again an unpredictable NextResultKey was returned, to be used in the next request |
| 4 | **There are no new Invalid Records from the /records pipeline of your integration. However, we query again (and again) until no invalid records are returned.**  **Request**  curl --request GET \  --url 'https://dig.geotab.com:443/invalidrecords?NextResultKey=51889&NumberOfResults=5' \  --header 'Authorization: Bearer eyJh--Bearer-token-clipped-TCsIA'  **Response**  {  "Error": [],  "Data": {  "NextResultKey": 51889,  "CurrentResultKey": 51889,  "TotalNumberOfInvalidRecords": 8,  "CurrentNumberOfInvalidRecords": 0,  "InvalidRecords": []  }  }  Comments:   * No invalid records were returned, so CurrentNumberOfInvalidRecords=0 * Still 8 invalid records exist for this account, so TotalNumberOfInvalidRecords=8 |
| 5 | **At this point, the integration pauses (waits) for 1 hour before checking again.** |
| 6 | **There are still no new Invalid Records from the /records pipeline of your integration, but it is time to check again.**  **Request**  curl --request GET \  --url 'https://dig.geotab.com:443/invalidrecords?NextResultKey=51889&NumberOfResults=5' \  --header 'Authorization: Bearer eyJh--Bearer-token-clipped-TCsIA'  **Response**  {  "Error": [],  "Data": {  "NextResultKey": 51889,  "CurrentResultKey": 51889,  "TotalNumberOfInvalidRecords": 8,  "CurrentNumberOfInvalidRecords": 0,  "InvalidRecords": []  }  }  Comments:   * No invalid records were returned, so CurrentNumberOfInvalidRecords=0 * Still 8 invalid records exist for this account, so TotalNumberOfInvalidRecords=8 |
| 7 | **The integration continues to query every hour.** |
| 8 | **Imagine there were no new invalid records for several days, and your invalid records have expired in DIG. Now there are no invalid records.**  **Request**  curl --request GET \  --url 'https://dig.geotab.com:443/invalidrecords?NextResultKey=51889&NumberOfResults=5' \  --header 'Authorization: Bearer eyJh--Bearer-token-clipped-TCsIA'  **Response**  {  "Error": [],  "Data": {  "NextResultKey": 51889,  "CurrentResultKey": 51889,  "TotalNumberOfInvalidRecords": 0,  "CurrentNumberOfInvalidRecords": 0,  "InvalidRecords": []  }  }  Comments:   * Back to no invalid records existing for this account, so TotalNumberOfInvalidRecords=0 |

# Best Practices & Usage Expectations

## /authentication/\*

Review how your integrations will implement the authentication workflows by understanding the [API Authentication Workflow Service Account Best Practices](https://docs.google.com/document/d/1aRPIDz7d49BEqEID_ZLjtrhwAuHacWO1WhTMUVUXwMI/edit#heading=h.84mdiwcgr1dj) as well as the [API Authentication and Token Management Implementation Strategies](https://docs.google.com/document/d/1aRPIDz7d49BEqEID_ZLjtrhwAuHacWO1WhTMUVUXwMI/edit#heading=h.ync3qm5aq8ef).

## Separation of the /records Pipeline & /invalidrecords Processes

Geotab recommends for simplicity that integrators consider the primary inline pipeline path of sending bulk records to DIG to be a separate process from the secondary system that queries periodically for invalid records.

This allows /records requests to “fire and forget” with high performance and scale using Fast ACK, when the response is 202 ACCEPTED. No further auditing or tracking from this portion of the integration would necessarily be required.

A separate secondary process can query **/invalidrecords** periodically to re-ingest records that were marked as invalid. These records should be saved to storage for human assessment and correction before resending those records for re-processing.

# Appendix

## Custom Telematics Device Records

This section outlines any supporting information for custom telematics device records that can be helpful beyond what the [OpenAPI yaml specification](https://docs.google.com/document/d/15uNuPqwFcPLe6vKs_JgY5nPTy2isQ3WYUu4oyQ3cEfQ/edit#heading=h.5rnqu38ynlfm) supplies.

Please adhere to this specification to align with record fields and their object models. To make requests against DIG, this document provides the [Records Workflow](https://docs.google.com/document/d/1XFHQ1s-um6HcW3qPRNiKX7bj-_X-O--4Fj4_j_An8U0/edit?ts=60ca2bc2#heading=h.opewb6fg3yvi) section above describing how to generally construct API requests to send records. The sections and samples below do not duplicate the record model specification nor iterate the matrix of possible sample requests.

The records workflow and the API specification should provide sufficient documentation to implement a custom telematics device integration with Geotab.

The sections below are additional documentation that goes beyond the specification and DIG API invocation. Where needed for specific record types, we will highlight scenarios and correlate with the MyGeotab platform.

### GpsRecord

**Representing your custom device on the live Map and rules-engine processing**

Most integrations desire the custom telematics device to indicate its current state to MyGeotab users (via live Map) or the MyGeotab application rules engine (to process exceptions).

The custom telematics device accomplishes that by manipulating the **isIgnitionOn** and **isGpsValid** properties at the same time. Below is a sample GpsRecord:

{"DateTime":"2020-01-01T01:00:00Z","SerialNo":"xxxxxxx","Type":"GpsRecord","IsGpsValid":True,"IsIgnitionOn":True,"Latitude": 43.641907, "Longitude": -79.455228, "Speed": 40}

To indicate that an asset has stopped moving (either at the end of a trip or the leg of a journey), the custom telematics device needs to send records in the following sequence, with its properties set as follows:

1. A record with isIgnitionOn=false and isGpsValid=true.
2. A record with (a) identical GPS coordinates as previous with (b) a timestamp difference >200 seconds.
3. A record with isIgnitionOn=false and isGpsValid=false to force the live Map to interpret a stop.

The custom telematics device will indicate that it is moving by sending a record with isIgnitionOn=true and isGpsValid=true.

| ✱ **NOTE:** When **isGpsValid** is set to **false**, the rules engine considers the GPS coordinates for this record as inaccurate and ignores them. By default, a GO device deems a coordinate as accurate if there is a GPS latch with at least five satellites, at which point **isGpsValid** is set to **true**. If there is less than that, the coordinates are still sent but with **isGpsValid**=**false,** which means the rules engine will possibly ignore them. |
| --- |

On the [Geotab live Map](https://docs.google.com/document/d/14GNbMq_ZKSUpkmSdJ8ws-DyO4P_HmTYOIu3k52f7KKY/edit#heading=h.xk2uv4qofarf), the only Iconography for vehicles states are:

* Square () - Vehicle is stopped outside a zone.
* Star () - Vehicle is stopped inside a zone.
* Triangle () - Vehicle is moving.

### StatusRecord and GenericStatusRecord

Both **StatusRecord** and **GenericStatusRecord** allow status data to be set to MyGeotab. The difference between the two is that **StatusRecords** map to legacy third-party source diagnostics, whereas GenericStatusRecords map to telematics device source diagnostics.

* To find the diagnostic code for **GenericStatusRecords**, please refer to [MyGeotab Diagnostics [PUBLIC]](https://docs.google.com/spreadsheets/u/0/d/1sy7IOhWmFoo40_I-ruOJO8bVTMLXqHa11d0oJtaoIcE/edit)
* To find the diagnostic code for **StatusRecords**, select **Diagnostics** from the **Engine and Device** tab in MyGeotab. Then, choose **Third-party** for the source filter.

In order for an integration to leverage status records, we recommend reviewing the MyGeotab diagnostics available, comparing and mapping to your system, experimenting/testing, and validating the expected behavior from the source system and ultimately how this is exposed within MyGeotab for your requirements.

| **✱ NOTE:** Geotab recommends using GenericStatusRecords, as **third-party source diagnostics will be deprecated in the future**. |
| --- |

### BinaryRecord: Storage of Arbitrary Binary/String Data

Consider the following sample of arbitrary data that an integrator would like to store in the Geotab ecosystem that falls outside the supported MyGeotab record data schemas. This can be unique per custom telematics device for a particular time and date.

This flexible binary record is useful as well for prototyping solutions integration systems to verify how data can leave and be retrieved.

**Data Sample**

Main Battery=12.89;Backup Battery=0;Box Serial Number=**######**;Firmware Version=0001\_tr00.SE7.G30.NN(Aug 1 2020 14:12:47)#Dis WxC:xC000002 GPS:S5-P4.0;Waypoints Usage=;SIM=XXXXXXX;Cell=;OTAB=N;

**Data Base64 Encoded**

TWFpbiBCYXR0ZXJ5PTEyLjg5O0JhY2t1cCBCYXR0ZXJ5PTA7Qm94IFNlcmlhbCBOdW1iZXI9IyMjIyMjO0Zpcm13YXJlIFZlcnNpb249MDAwMV90cjAwLlNFNy5HMzAuTk4oQXVnICAxIDIwMjAgMTQ6MTI6NDcpI0RpcyBXeEM6eEMwMDAwMDIgR1BTOlM1LVA0LjA7V2F5cG9pbnRzIFVzYWdlPTtTSU09WFhYWFhYWDtDZWxsPTtPVEFCPU47Cg==

**DIG API Request**

This data can be sent using the **BinaryRecord** record type; set the “Data” field to the arbitrary data, Base64 encoded.

**Retrieving the Binary Data**

This data can be retrieved from MyGeotab using the [MyGeotab SDK](https://geotab.github.io/sdk/) using [GetFeed](https://geotab.github.io/sdk/software/guides/data-feed/) as shown below:

MyGeotab API Request

| api.call("GetFeed", {  "typeName": "**BinaryData**",  "resultsLimit": 10,  "fromVersion": "0000000000000000",  "search": {  "deviceSearch": {  "id": "**XXXXX**"  }  }  }, function(result) {  console.log("Done: ", result);  }, function(e) {  console.error("Failed:", e);  }); |
| --- |

MyGeotab API Response

| Object { ... }  -data: Array[1]  -0: Object  binaryType: "**ThirdPartyData**"  controller: "ControllerNoneId"  data: "**TWFpbiBCYXR0ZXJ5PTEyLj(snipped)WxsPTtPVEFCPU47Cg==**"  dateTime: "2020-10-27T19:00:00.000Z"  -device: Object  id: "**XXXXX**"  id: "a7A7zKgoMRrfx2Mv0EeVdAA"  version: "0000000000100de7"  length: 1  toVersion: "0000000000100de7" |
| --- |

### DriverChangeRecord: Driver Assignment

This functionality is equivalent to the [Driver Assignment feature in MyGeotab](https://community.geotab.com/s/article/How-to-Assign-Drivers-to-a-Vehicle) and similar to the [blogged NFC Driver ID scenario](https://www.geotab.com/blog/driver-id/).

For custom telematics device integrations, we can consider the goal of sending a “Driver Assignment” event from a key swipe or other custom hardware or driver operation.

| ✱ NOTE: In order to use this feature, it is expected that the Driver ID key value is configured within MyGeotab. See section “**How to Assign a Driver ID Key to a Driver in MyGeotab**” [here.](https://www.geotab.com/blog/driver-id/) |
| --- |

**DIG API Request**

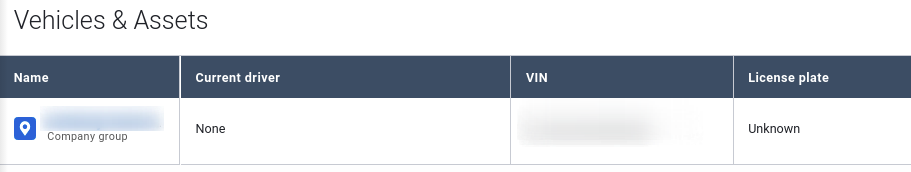
Integrators can use the **DriverChangeRecord** for driver assignment.

**Supported Key Types**

| **Key Type Value** | **Key Type Label MyGeotab Mapping** | **Driver ID** |
| --- | --- | --- |
| 84 | Custom | Unique string, Base64 encoded |

**MyGeotab UI Outcomes**

Before (no driver associated)



After (a new driver is assigned)



### Fault Records

There are different SAE protocols for fault codes depending on the age and type of vehicle.

The following record types enable the ingestion of fault data records into the Geotab platform via DIG.

| **Geotab Fault Record Name** | **Standards & Fault Codes** | **Supplementary Notes** |
| --- | --- | --- |
| OBD II Fault Record | [OBD II Codes Defined](https://www.obd-codes.com/faq/obd2-codes-explained.php)  [OBD II Trouble Codes](https://www.obd-codes.com/trouble_codes/) | [OBD 2 Explained](https://www.csselectronics.com/screen/page/simple-intro-obd2-explained/language/en) |
| J1939 Fault Record | [SAE](https://www.sae.org/standardsdev/groundvehicle/j1939a.htm) |  |
| J1708 Fault Record | [SAE](https://www.sae.org/standards/content/j1708_200408/) |  |
| Generic Fault Record | N/A - Geotab’s generic fault record for proprietary implementations | Please [contact integration solutions](https://docs.google.com/document/d/15uNuPqwFcPLe6vKs_JgY5nPTy2isQ3WYUu4oyQ3cEfQ/edit#heading=h.hzkgpqwq4aj6) for support. |

There are different input criteria based on the protocol utilized by the connected vehicle, so integrators should utilize as appropriate. Faults captured via these record types are visible by navigating to [Engine Faults on MyGeotab’s left menu.](https://www.geotab.com/video/how-to-view-engine-faults-in-mygeotab/)

Each record model and its properties are denoted in the DIG API on Github; <https://github.com/Geotab/data-intake-gateway>.

Further context and specific explanations are provided below:

* Default State Values:
  + Fault and Lamp States default to false.
* Fault State:
  + Translation is mapped to these [fault states in the MyGeotab SDK](https://geotab.github.io/sdk/software/api/reference/#T:Geotab.Checkmate.ObjectModel.Engine.FaultState)
  + Currently MyGeotab supports if a fault state is active or not active.
* Lamp State:
  + Translation is mapped to these [lamp states in the MyGeotab SDK](https://geotab.github.io/sdk/software/api/reference/#T:Geotab.Checkmate.ObjectModel.Engine.FaultLampState)
* Failure Mode Identifier:
  + Translation is mapped to the [“code” (integer) in the MyGeotab SDK](https://docs.google.com/document/d/1yJhpfazIRV8ZFM7MFzk_0HyZAAZhizKF4FkuJ-xJ84A/edit#)