
Project:	Geotechnical and Drainage Management Service		
Our reference:	n/a	Your reference:	n/a
Prepared by:	Chloe Spurling	Date:	October 2024
Approved by:	Matt Lane	Checked by:	Keith Halstead
Subject:	REST API documentation		

This document is issued for the party which commissioned it and for specific purposes connected with the above-captioned project only. It should not be relied upon by any other party or used for any other purpose.

We accept no responsibility for the consequences of this document being relied upon by any other party, or being used for any other purpose, or containing any error or omission which is due to an error or omission in data supplied to us by other parties.

This document contains confidential information and proprietary intellectual property. It should not be shown to other parties without consent from us and from the party which commissioned it.

1 Introduction

1.1 Scope of document

This document includes a relatively high-level description of the REST APIs used by the GDMS system. It provides a general overview to enable users to start using the APIs to retrieve GDMS data, including worked examples covering the main endpoints and different ways of retrieving data, but does not describe each endpoint.

All endpoints are included in the Swagger JSON format files available from GDMS support. At present only the main endpoints that are used to retrieve records are specifically documented by way of a summary within these JSON files. Endpoints that allow data to be changed are not currently documented and are not intended for third-party use at present.

1.2 Document conventions

Throughout this document URLs and snippets of JSON data or code are shown.

Endpoint URLs are preceded by their HTTP method (e.g. GET or POST) and where part of a URL is a variable or parameter that needs to have a suitable value, then {curly braces} are used. For example:

GET <https://api.gdms.assetia.cloud/Geotechnical/Assets/{assetId}>

Where JSON or code is included in the document, a fixed width font is used. Within this, text in *italics* provides a description of what would be included, such as the type of data or to indicate where example data is shortened for brevity. Any curly braces are part of the sample JSON:

```
{
  "ID": GUID as a string value,
  "height": number,
  "description": string,
  other properties omitted
}
```

1.3 What is a REST API?

- REST = Representational State Transfer
- API = Application Programming Interface

An API is a means to allow communication between software programs or systems. A web API is specifically a means for a web server to externally send and receive data, such as to allow a website or mobile application to retrieve and display data from a remote web server. Web APIs may be intended only for use by the software written by the operator of the API, or may be available for third party use. Most APIs will require some form of authentication to control access, at application level and/or user level. Some web APIs allow limited free use but require payment for larger numbers of requests, for commercial use of the data or to access more valuable data.

Some examples of web APIs that can be used by third parties include:

- Network Rail APIs for real-time running of trains, displayed by websites such as OpenTrainTimes
- Met Office weather APIs to allow weather forecast and observation information to be extracted and used by third parties
- Spotify APIs to allow the Spotify music catalogue to be browsed and searched by third party software
- What3words API to allow third party developers to convert between a geographical location and a what3words text string (e.g. "beans.again.voting").

Web APIs expose “endpoints” which are web addresses used for a specific type of request or resource. For example, the Spotify API has an endpoint that provides information about an artist, another endpoint that provides information about an album, and other endpoints that allow searching of data. Web APIs that allow data on the server to be changed will have separate endpoints depending on whether data is being requested or sent.

Each valid request to an API endpoint will result in one response which, depending on the nature of the request, may be a single record, multiple records, or just a response code indicating success or otherwise.

A request may also include data sent to the server in the following ways, as defined by the API endpoint:

- HTTP headers, particularly to include authorisation tokens or to specify the data format required (e.g. JSON or XML, where the endpoint supports this)
- within the URL, e.g. query criteria:
 - <https://api.myassetsystem.com/assets?id=123>
 - <https://api.myassetsystem.com/assets/123>
- in the “payload” or body of the request, e.g. complex query criteria, a new record, binary data

REST is a software architecture for communications across the web, that sets out principles that a “RESTful API” is expected to comply with, such as:

- separation of the client software / user interface from the server software
- a layered server architecture that is transparent to the client but allows scalability (e.g. by having a single gateway for the client to interface with, that may be underlain by numerous load-balanced servers to manage demand)
- proper use of methods defined in the HTTP standard (e.g. GET to request data, POST to send data for the server to process).

REST is not considered a standard but uses other standards such as HTTP (to define the exact structures of requests and responses, including response codes such as “200” for successful) and JSON or XML standards to define the format of data transferred.

REST APIs may also implement standards such as OAuth for authorisation and OData for the structure of requests and responses.

1.4 What is JSON?

- JSON = JavaScript Object Notation

JSON is a text-based, open standard file format for data interchange that is compatible with the JavaScript programming language. Data consists of name-value pairs of properties. It is particularly used by web APIs for transferring data, as an alternative to XML. Relevant standards are ECMA-404 and ISO/IEC 21778:2017.

An example of JSON data representing the details of a fictional person is shown below:¹

```
{
  "person": {
    "firstName": "John",
    "lastName": "Smith",
    "age": 27,
    "isMarried": true,
    "address": null,
    "phoneNumbers": [
      {
        "type": "home",
        "number": "01632 123456"
      }, {
        "type": "office",
        "number": "01632 987654"
      }
    ]
  }
}
```

The example above comprises a “person” object with the properties: “firstName” (string), “lastName” (string), “age” (number), “isMarried” (boolean), “address” (with no value) and “phoneNumbers”. “phoneNumbers” is an array containing two values, each of which is an object with a “type” and “number” property. These terms are explained further below.

Each property’s name and value are separated by a colon, and each property is separated from the next by a comma. Property names are “double-quoted” and are conventionally (but not necessarily) lowerCamelCase. It is not possible to include anything that is not part of the name-value data structure, such as JavaScript comments.

Basic (“primitive”) data types of property values include:

- strings² (e.g. “John”)
- numbers³ (e.g. 27)
- booleans (true or false).

Other data types such as GUIDs or dates would be represented by a string (e.g. “2022-03-11”) or a number (e.g. milliseconds since epoch time). Any property that has no value must have the word `null`.

Where one or usually multiple sub-properties define the value of a single property (e.g. a “person” in the example above), these are an object enclosed with {curly braces}.

Where a single property can have multiple values of the same type, these values are stored as an array enclosed with [square brackets]. Arrays may either contain primitive values (e.g. an array of numbers such as [1,3,5,7]) or objects such as “phoneNumbers” in the example. Each value within an array is normally (but not necessarily) of the same type, and if they are objects are normally the same type of object with the same properties. Where a JSON property may have multiple values, it will always be represented as an array even if it currently contains only one value (e.g. [42]) or no values (i.e. []), to retain the same JSON data structure.

¹ Based on example at <https://en.wikipedia.org/wiki/JSON>. Phone numbers are from Ofcom’s range of numbers reserved for TV/film.

² JSON strings can be of any length.

³ JSON does not differentiate between types of numbers, e.g. integers and floating point.

The JSON itself must also either be an object or an array, depending on the nature of its top-level data. In the example above it is an object with a single “person” property, and therefore the JSON is enclosed with {curly braces}. If the JSON had only included the array of phone numbers, it would have been:

```
[
  {
    "type": "home",
    "number": "01632 123456"
  },
  {
    "type": "office",
    "number": "01632 987654"
  }
]
```

White space is only significant in JSON within double quotes, so the JSON below is equivalent to above:

```
{"person":{"firstName":"John","lastName":"Smith","age":27,"married":true,"address":null,"phoneNumbers":[{"type":"home","number":"01632 123456"}, {"type":"office","number":"01632 987654"}]}}
```

1.5 How does GDMS implement web APIs?

The GDMS system has REST API endpoints for all actions that retrieve or update data in the system. This document explains how to access the GDMS APIs and use them to retrieve data. Use of the API endpoints for updating data is not permitted at present, except by use of the web and mobile applications provided by the GDMS team.

The GDMS web application is a JavaScript application that runs within a web browser. This application communicates with the GDMS servers using the APIs described in this document. Before attempting to use any of the GDMS APIs it is recommended to observe the network activity within a web browser, while browsing similar data to what you require. This will assist with identifying appropriate endpoints, and understanding what is included with the request and in the response. Your requests should match those made by GDMS as closely as possible to ensure the response is as intended; some requests may include default properties which may appear unnecessary but if omitted altogether could lead to an invalid request or unintended response.

In Chrome and Edge browsers this activity can be viewed in the Developer Tools window (press CTRL+SHIFT+I when a GDMS browser tab is open). Then browse to the “Network” tab, filtered to “Fetch/XHR” requests. This document does not further describe how to use the Developer Tools window; a link to full documentation provided by your browser vendor is available within the “Help” menu.

This document does not describe the functionality of GDMS. Further information can be found at <https://help.gdms.assetia.cloud>.

1.6 Note on worked examples

The worked examples attempt to cover all the different ways of requesting data, and all types of data, but not all combinations of these. The principles covered in one example will be applicable to other types of data, and this is usually explained, including any notable exceptions. The examples given mostly reflect the kind of queries that could be required of GDMS data, but are also intended to illustrate how to formulate a request.

Numeric IDs and GUIDs are used in the examples but, in most cases, these are fictional and will not return data if used as-is. Some figures and references shown in the example results are also fictionalised.

2 General information

2.1 Authentication and authorisation

2.1.1 General access

Prior to being able to access any GDMS endpoint, authentication must be carried out using an existing GDMS account. GDMS accounts are allocated to named individuals by the GDMS support team. Usernames must be a valid email address that the named individual can access, and passwords must be set by that individual and meet certain complexity requirements.

GDMS access permissions are defined per module, per Area (for modules that contain Area-specific data) and at one of the following levels, which can vary between modules and/or Areas:

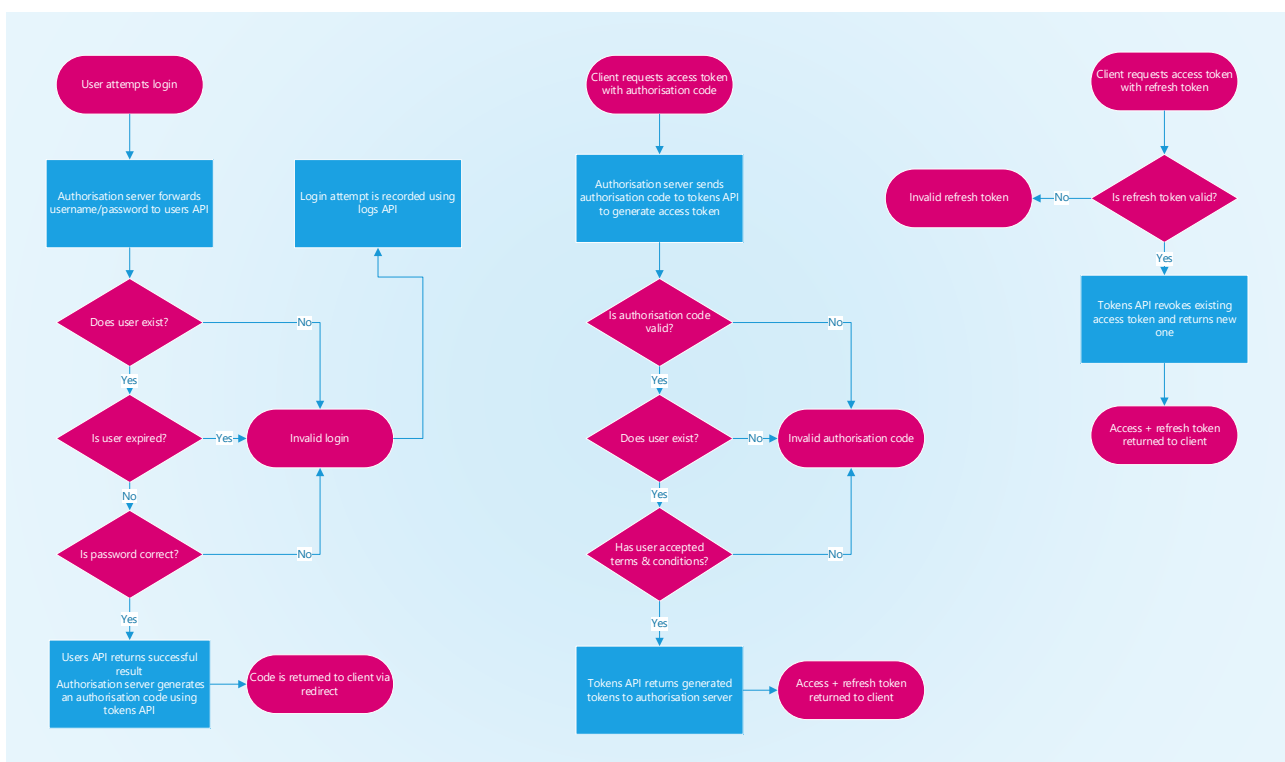
- Default (view-only)
- Edit (Default plus ability to edit most data)
- Manage (Default plus Edit plus ability to carry out actions such as approving or locking data and editing locked down data that Edit users cannot)
- Review (Default plus occasionally ability to carry out a special action or the access level is used to receive notifications or access dashboards)

Access permissions for a user account are the same whether the user accesses GDMS via the Assetia website or via the APIs.

2.1.2 Authentication process

2.1.2.1 Process overview diagram

The diagram below provides an overview of the process for authenticating with GDMS, and obtaining and refreshing authorisation tokens.



2.1.2.2 Process description

All authentication with GDMS must be carried out by opening the following website and allowing the user to login. This interaction with the system cannot be completed via an API endpoint or by using your own login form. The user must also have previously accessed GDMS and accepted its terms and conditions.

https://auth.gdms.assetia.cloud/login?client_id={clientId}&redirect_uri={redirectURL}

- {clientId} must be replaced by a value to identify your application, which can be obtained from GDMS support
- {redirectURL} must be replaced by a URL that you wish to redirect to after the user has successfully logged in⁴

After the user has logged in, the web browser will redirect to the URL specified. In addition, a “code” parameter will be added to the redirection URL, i.e. “{redirectURL}?code={code}”. This code must be used by your application to request authentication tokens from GDMS within 60 seconds of the user logging in.

To obtain authentication tokens, a POST request must be sent to <https://auth.gdms.assetia.cloud/token> with the following as application/x-www-form-urlencoded data in the request body:

*code: value that was returned in the “code” URL parameter after logging in,
 client_id: your application’s identification code, as was used to access the Login form,
 grant_type: authorization_code,
 redirect_uri: your application’s redirection URL, as was used to access the Login form, with URL-encoding⁵*

For example:

`code={code}&client_id={clientId}&grant_type=authorization_code&redirect_uri=https%3A%2F%2Fyoursite.com%2Floggedin`

If the request is valid, the following JSON is returned:

```
{
  "access_token": string value for the access token
  "expires_in": number of seconds after which access_token expires (currently 3600, i.e. 1 hour)
  "refresh_token": string value for a token that allows the access token to be renewed
  "token_type": "Bearer"
}
```

All subsequent API requests to GDMS must include the following in the header:

Authorization: Bearer {access_token}

When the access token expires, API requests will return a 401 status code with the message “Authentication token is invalid or expired”. To renew the access token, you must send a POST request to <https://auth.gdms.assetia.cloud/token/refresh> with the following as application/x-www-form-urlencoded data in the request body:

*access_token: the value of the access_token that is being replaced
 refresh_token: the value of the refresh_token that was assigned after initial authentication
 grant_type: refresh_token*

The above request returns data in the same format as that used for the initial access token. The “refresh_token” will remain the same but the “access_token” will be different.

⁴ For a web application, {redirectURL} will be the URL of your application that you wish the user to be sent to, e.g. “https://yoursite.com/loggedin”. For a native application the {redirectURL} can be something like “yourappname://callback”.

⁵ A URL-encoded version of “https://yoursite.com/loggedin” is “https%3A%2F%2Fyoursite.com%2Floggedin”.

Refresh tokens currently expire after 7 days. After this time, users must repeat the login process to continue accessing the APIs.

If a user's access is disabled on the system after obtaining authentication tokens, then access may be possible for at most 1 hour until their current access_token expires. It will not be possible for them to refresh the token or login again.

2.1.3 Authenticated user information

The Users API is not included in the documentation. The following endpoints can be used to return the currently authenticated user's ID, details and permissions, and to interpret the module and level GUIDs in the permissions data:

- GET <https://api.gdms.assetia.cloud/users/current>
- GET <https://api.gdms.assetia.cloud/users/{userId}/permissions>
 - where {userId} is the GUID "id" property returned by /users/current
- GET <https://api.gdms.assetia.cloud/permissions/modules>
- GET <https://api.gdms.assetia.cloud/permissions/levels>

Some of a user's permissions may be associated with an Area ID. Area details are covered in the Network Model and Locations API (see section 5).

If the user's organisation details are required, these can be obtained with:

- GET <https://api.gdms.assetia.cloud/users/organisations/{organisationId}>
 - where {organisationId} is the GUID "organisation" property returned by /users/current

If a user needs to update their information, then they must do this through the GDMS website.

Please note that it is only possible to obtain any user-related data through the API that the authenticated user could access within the GDMS website. In particular, users can choose to make their details visible or invisible to users in other organisations, and this applies both through the website and API.

2.2 Data formats

With very few exceptions, GDMS API endpoints receive and return data in JSON format. The system does not support XML for requests or responses.

The returned JSON will only include ASCII text, and endpoints typically only return one type of object (either a single record or an array of multiple records of the same type). Endpoints do not return nested data (e.g. an asset record with its inventory items nested).

Some endpoints return map geometry data which is included within the JSON as a Well-known text (WKT) string value, to Ordnance Survey (OS) grid (EPSG:27700). These will define a point, line or polygon for each record, as appropriate to the type of record. Records with a point location may also include OS eastings and northings as numeric properties.

Some endpoints return data in CSV format, equivalent to outputting a GDMS summary grid to CSV. There is typically one endpoint for each of the main object types that will do this.

The Files API also includes an endpoint that will return a file in its original format.

2.3 URL structure

Most commonly used endpoint URLs follow one of the formats below. All GDMS URLs are case-insensitive. Endpoints currently return only one principal type of record, rather than nesting related/child records within a parent record, requiring separate requests for the related records. All requests must use HTTPS.

- <https://api.gdms.assetia.cloud/{module}/{recordType}/{action}>
 - typically a GET or POST request for multiple records of recordType, with “action” being something like “query”, “summary” or “csv”
- <https://api.gdms.assetia.cloud/{module}/{recordType}/{recordId}>
 - typically a GET request to return a single record by its unique ID, or a PATCH request to update it
- <https://api.gdms.assetia.cloud/{module}/{recordType}/{recordId}/{action}>
 - typically a POST or DELETE request to update or carry out an action on a single record by its unique ID

There are some differences between modules, and these are described in further detail in the relevant section of this document, along with the main “recordType” values.

2.4 Unique IDs

All records in GDMS are uniquely identified by a globally unique identifier (GUID). GUIDs are alphanumeric strings that are generated by an algorithm which is deemed to be reliably unique worldwide, whoever generates it and however often.⁶ These GUIDs are used not only as the “Id” of a record, but also as primary and foreign keys to link related records across tables. GUIDs are unique not just for all records of a particular type, but across all types of records.

Most of the principal types of records in GDMS also have a numeric “visible ID”, which is typically presented to users as the “ID”. These are generated in numeric sequence of when a record is initially created within the database, and can only be allocated by GDMS. Numeric IDs are unique for a particular type of record, but not across different types of records. Numeric IDs are not used as keys to link records together, and therefore endpoints that return child records will include the parent record’s GUID but not the parent record’s numeric ID.⁷

Both the GUID and numeric ID remain unchanged for a record throughout its lifetime, e.g. during edits.

2.4.1 Returning a record based on numeric ID

All API endpoints for specific records require the record’s GUID. If the GUID for a record is not known but the numeric ID is, then an appropriate endpoint that can return multiple records should be used, filtered by the numeric ID value. Such endpoints usually return most if not all of the data fields for the record, so further requests may not be needed, but the GUID is included to allow requests for the full record if required. In the later sections of this document, some worked examples are provided.

2.5 Picklist fields

Where possible, text fields in GDMS have pre-defined options as opposed to being free text, to maintain consistency within the data. These fields include those that a user would edit using a picklist (e.g. to select the type of an asset) and some that are determined by the system as part of a workflow (e.g. the status of a record).

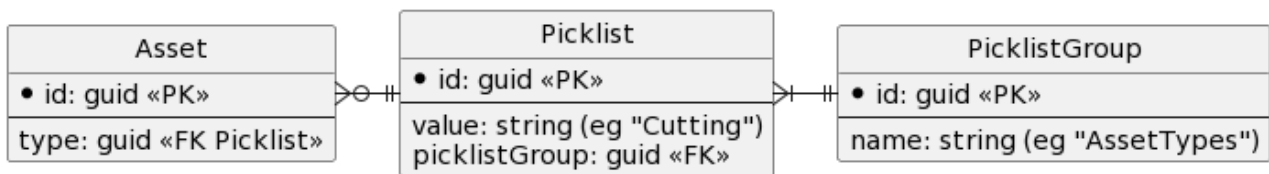
⁶ One definition is that if every human each generated 600,000,000 GUIDs there would be a 50% probability of a duplicate. This is not universally unique but within the context of GDMS is effectively guaranteed to be unique throughout the system.

⁷ For example an inventory item includes the GUID of its parent asset, but not the numeric ID of the asset.

The data for these fields is managed as follows, taking as an example the “type” attribute of an asset:

- Each module’s picklist data is stored within that module, separate from other modules.
- The “type” field is stored in the asset record as a GUID. The API passes GUIDs as string properties.
- The GUID equates to a picklist ID, which maps to a value (e.g. “Cutting”) and a picklist group (e.g. “AssetTypes”) to determine the picklist entry as an option for a particular group.
 - Picklist values are always treated as strings, even if their value is numeric.
- The picklist group ID is a GUID that can be used to retrieve all applicable picklist entries for that group, and the name of the group.

The picklist data structure for an asset of type “Cutting” is shown below:



Some picklist values could be present in multiple picklist groups; for example “2” is valid as an asset inspection return period and as a condition set classification. In this case, there are separate picklist entries for each of the groups, each with a different picklist ID that will be for the entry in the applicable picklist group.

API endpoints for picklist information vary by module but typically include:

- Named endpoints to return all of the available picklist entries for one group, e.g.:
 - GET <https://api.gdms.assetia.cloud/geotechnical/assettypes>
 - GET <https://api.gdms.assetia.cloud/reports/picklists/types>
- Some modules have endpoints to return a list of all picklist entries and picklist groups, e.g.:
 - GET <https://api.gdms.assetia.cloud/geotechnical/picklists>
 - GET <https://api.gdms.assetia.cloud/geotechnical/picklistgroups>

Picklist and picklist group data does not frequently change, and your application may therefore wish to cache such data for a session.

2.6 Defining requests for multiple records

2.6.1 Overview

For each type of record in GDMS, one or more of the following POST endpoints are available that can return one or more, or even all, of the records of that type:

- “query” is available for almost any type of record and typically returns data in its raw and complete format. Picklist field criteria will usually need to be provided as GUIDs rather than readable values.
- “summary” is available in parallel with the “summary grids” on GDMS and typically returns data in a more human-readable format. Picklist fields may be passed as values rather than GUIDs. Some additional useful fields derived from related records may be included such as length, where this isn’t part of the raw stored data.
- “csv” normally returns similar data to “summary” but in CSV format rather than JSON and without any pagination. CSV files comply with RFC 4180 and the first line will contain field headings.

As described below, these requests may include a JSON object to sort and/or filter the records that the API will return. To help understanding, the parts of this object are equated to the WHERE, ORDER BY, OFFSET

and FETCH statements within a SQL query. Some specific requests allow additional fields to be included, as defined in the Swagger JSON files. The general structure of the JSON object is:

```
{
  "offset": number,
  "limit": number,
  "filter": {object},
  "sort": [array of objects]
}
```

Each of the four properties is optional and they are described in the following sub-sections.

In addition, some specific endpoints support additional properties, for example to query all of the exploratory locations for a specific geodataset ID or within a geographical area (see section 2.6.7), or to include or exclude archived GAD records (see section 6.3.6). Some examples of these are given in later chapters.

This document has a relatively brief description, and it is recommended to use the summary grids in GDMS to replicate the sorting or filtering you require to see how the request is generated, in conjunction with the output in your web browser's Developer Tools window or similar.

In addition to the POST endpoints some record types have GET endpoints that allow simple URL parameters such as "offset" and "limit". It is not practical to use these with the more complex "filter" and "sort" parameters.

2.6.2 Response

Query and Summary endpoints will return results as follows:

```
{
  "results": array of records returned by the request,
  "count": total number of records matched by the filter criteria ignoring offset and limit,
  "total": total number of records that would be returned without any filter criteria
}
```

The number of records returned in the "results" array is not necessarily the same as the "count" or "total". That number can be determined in JavaScript simply as `results.length`.

CSV endpoints will return just the records without any other information.

2.6.3 No criteria

To return all records of a particular type with no filtering or sorting, then an empty JSON object { } must be passed if the endpoint requires a body.

Such requests should not generally be made if they would return a large amount of data. Users should discuss their requirements with GDMS support to ensure that this would not have an adverse impact on the system or other users.

2.6.4 Offset and limit

On GDMS, grid outputs are usually paginated to 25 records, by using the following in the request:

- `offset`: number of records to offset, where 0 is default if unspecified and means no offset (i.e. the first record is returned).
- `limit`: the number of records to return (e.g. 25) starting at any offset. If unspecified then all records starting at any offset are returned.

Offset and limit are applied to the results after any filtering and sorting are applied.

The following returns the first 25 records, equivalent to the SQL: `OFFSET 0 ROWS FETCH NEXT 25 ROWS ONLY`

```
{
  "offset": 0,
  "limit": 25
}
```

If any filtering and offset means that fewer records remain to be returned than the “limit”, then only those remaining records are returned rather than a full “page”. For example, if there are 104 records, with an offset of 100 and a limit of 25, then just 4 records would be returned (records 101–104).

Please note that offset does mean offset from the start, hence an offset of 1 would omit the first result.

2.6.5 Filter criteria

The filter object allows complex filtering of the records by multiple fields and by multiple criteria on a field. If not included then no filtering is applied, and all records could be returned.

2.6.5.1 Single criterion

For a very simple filter with a single criterion on a single field, such as to find a record with a particular numeric visible ID, the following is sufficient in most modules:

```
{
  "filter": {
    "filters": [
      {
        "field": "visibleId",
        "not": false,
        "operator": "Equals",
        "value": 12345
      }
    ]
  }
}
```

Although there is only one filter criterion in the above, because “filters” can have multiple criteria in more complex cases (see section 2.6.5.2), it must still be provided as an array.

In this simple example, the following parts of the request would be changed as required:

- “field”: the name of the field that is being filtered
- “not”: true or false, working in conjunction with “operator”. It is useful to read this as a double negative.
- “operator”: one of the following, as applicable to the type of field:
 - “LessThan”: numeric or date/time fields
 - “GreaterThan”: numeric or date/time fields
 - “Equals”: any field type
 - “Contains”: string fields
 - “StartsWith”: string fields
 - “EndsWith”: string fields

- “value”: the value that is being used as the filter criterion. The data type of “value” must match the data type of “field”, i.e. numeric, string or boolean.⁸ If the `visibleId` field was a string, then the value would need to be enclosed in quotes, e.g. “12345”.

The available operators do not specifically include criteria such as “Less Than Or Equal To”, which is instead achieved as “Not Greater Than”:

```
"not": true
"operator": "GreaterThan"
```

2.6.5.2 Multiple criteria

The example below returns records that are of type “Embankment” and are on either the “A30” or “A38” roads. It is equivalent to the SQL: `WHERE type = 'Embankment' AND (road = 'A30' OR road = 'A38')`

```
{
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "OR",
        "filterGroups": [],
        "filters": [
          {
            "field": "type",
            "not": false,
            "operator": "Equals",
            "value": "Embankment"
          }
        ]
      },
      {
        "condition": "OR",
        "filterGroups": [],
        "filters": [
          {
            "field": "road",
            "not": false,
            "operator": "Equals",
            "value": "A30"
          },
          {
            "field": "road",
            "not": false,
            "operator": "Equals",
            "value": "A38"
          }
        ]
      }
    ]
  }
},
```

⁸ The appropriate field type may be determined from the Swagger JSON documentation, or by using the GDMS user interface to filter similar data and reviewing the requests sent.

```

    "filters": []
  }
}

```

The filter object is structured as follows:

- “condition”: either “AND” or “OR”, refers to the criteria in the sibling filterGroups or filters array
- “filterGroups”: an array of objects that each define one field’s filter criteria. The sibling “condition” is applied when there are multiple filterGroups.
- “filters”: an array of objects that each define one filter criterion on one “field”. The sibling “condition” is applied when there are multiple filters. The four properties of each object within the “filters” array are described in section 2.6.5.1.

2.6.6 Sort order

The records can be sorted using the sort property prior to the API endpoint returning the results. If any offset or limit is specified, then these apply to the sorted records.

The GDMS user interface only allows sorting by one field, ascending or descending. The API permits sorting by multiple fields in combination, equivalent to a SQL “WHERE” statement. All types of fields are supported, but string fields will sort alphabetically even if they contain numbers (e.g. “1”, “10”, “11”, “2”, “3”, etc).

The example below sorts records by “finalDepth” with the highest values first (descending order), followed by “visibleId” in ascending order. It is equivalent to the SQL: ORDER BY finalDepth DESC, visibleID ASC:

```

{
  "sort": [
    {
      "direction": "desc",
      "field": "finalDepth"
    },
    {
      "direction": "asc",
      "field": "visibleId"
    }
  ]
}

```

The “sort” property must be provided as an array, even if sorting by only one field. The fields are sorted in the order they appear within the “sort” array.

As in SQL, ascending (“asc”) order is implied by default, and the “direction” property may be omitted in this case.

If sort order is not specified, then the returned results should be assumed to be in random order. In many cases the server does apply a default sort order (e.g. numeric visible ID), but this should not be relied upon and is primarily to suit the GDMS user interface. Sorting is highly recommended if offset and/or limit are to be used; otherwise, it may be appropriate to sort the results within your own application.

2.6.7 Geographical criteria using Well-Known Text (WKT) polygons

Some geographically located records in GDMS support requests with location criteria. This may be included in the request instead of or in addition to filtering, sorting and/or paginating the response.

For example:

```
{
  "searchGeometry": "POLYGON ((534500 153000, 535500 153000, 535500 152500, 534500 152500, 534500 153000))"
}
```

Although the location being used in this example is a simple north-oriented rectangle, any WKT polygon may be defined. Note that the polygon must be closed so for a rectangle there are five pairs of coordinates and the last coordinate pair ("534500 153000") is the same as the first. All coordinates must be to OS grid.

This approach is currently supported for the following types of records:

- geotechnical asset data:
 - geotechnical assets
 - inventory items
 - inventory sets
 - condition items
 - condition sets
- exploratory locations
- reports

Where other types of records include easting and northing fields in responses, then these can be used within filter criteria, e.g. to define a rectangular bounding box.

An example of a geographical search using both methods is given in section 4.3.3.

The "count" and "total" numbers in the response will be as follows:

"count": *number of records within the geographical limits considering any additional filter criteria but ignoring offset and limit,*

"total": *total number of records within the geographical limits ignoring any additional criteria*

2.6.8 Request number of records only

To request the number of records that would be returned without returning any record data, pass a limit of 0 in the request body:

```
{
  "filter": as required,
  "offset": 0,
  "limit": 0
}
```

The "count" value in the response will show the total number of results meeting your filter criteria.

Only one set of filter criteria can be passed per request, and therefore if multiple counts are required (e.g. the number of each type of asset) then a separate request is needed for each.

3 Reports Archive API

Full Swagger documentation for the Reports Archive API is included in the accompanying Reports.json file.

3.1 Principal entities

The Reports Archive module contains the following principal entity:

- Report

3.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/reports/{id}>
 - return information for a single report where {id} is the report's GUID
- POST <https://api.gdms.assetia.cloud/reports/{action}>
 - return information for multiple reports where {action} is "query", "search" or "csv"
- GET <https://api.gdms.assetia.cloud/reports/picklists/{picklistGroup}>
 - return the permitted values and IDs for a picklist field, where {picklistGroup} identifies the field, e.g. "types", "statuses"
- GET <https://api.gdms.assetia.cloud/reports/{id}/{data}>
 - return related data for a single report where {id} is the report's GUID and {data} is the type of information, e.g. related roads or PDF files/bookmarks
- POST <https://api.gdms.assetia.cloud/reports/{id}/{action}>
 - (not supported) actions that change data for an individual report where {id} is the report's GUID

The ability to download files such as a report's PDF package or certificate is covered in the Files API (see section **Error! Reference source not found.**).

3.3 Worked examples

3.3.1 Return metadata for a report where numeric ID only is known

Example: return data for report 123

As the report GUID is not known, it is necessary to make a filtered request to a multiple results endpoint.

Endpoint: POST <https://api.gdms.assetia.cloud/Reports/query>

As there is only one criterion, the shortened version of the "filter" payload can be used (section 2.6.5.1):

```
{
  "filter": {
    "filters": [
      {
        "field": "visibleId",
        "not": false,
        "operator": "Equals",
        "value": 123
      }
    ]
  }
}
```


As there is only one report with a numeric ID of 123, provided this report exists, the endpoint will return a results array containing one report's data. This endpoint returns data including human-readable picklist values as follows (shortened):

```
{
  "results": [
    {
      "id": "29ebaccd-246d-4263-b835-1d8e48841599",
      "visibleId": 123,
      "title": " M1234 Principal Inspections Report 2021/22",
      "type": " Principal Inspection Report",
      other report properties...
    }
  ],
  "count": 1,
  "total": 19792
}
```

If other data for the report is needed, then the report's GUID can be retrieved as `results[0].id` and passed into an appropriate endpoint.

3.3.2 Return details of one report with a known GUID and determine its report type

Example: return data for report ID "29ebaccd-246d-4263-b835-1d8e48841599" and determine its report type.

As the report's GUID is known, the specific report can be requested.

Endpoint: GET <https://api.gdms.assetia.cloud/reports/29ebaccd-246d-4263-b835-1d8e48841599>

If the report exists, it will return data as follows (shortened):

```
{
  "id": "29ebaccd-246d-4263-b835-1d8e48841599",
  "visibleId": 123,
  "type": "d4825b12-48ae-415f-9080-c421c2321ace",
  "title": "M1234 Principal Inspections Report 2021/22",
  other report properties...
}
```

The type of the report is given by the GUID in the "type" property. This now needs to be converted to its equivalent human-readable picklist value.

Endpoint: GET <https://api.gdms.assetia.cloud/reports/picklists/types>

This will return an array of all report types as follows (shortened):

```
[
  other picklist entries...,
  {
    "id": "a2f510e6-f38e-4604-a535-67d234441889",
    "picklistGroup": "618882f8-1b58-46df-9b91-d46fe69d0525",
    "value": "Preliminary Studies (Scoping/Options)",
    "description": ""
  },
]
```

```
{
  "id": "d4825b12-48ae-415f-9080-c421c2321ace",
  "picklistGroup": "618882f8-1b58-46df-9b91-d46fe69d0525",
  "value": "Principal Inspection Report",
  "description": ""
},
  other picklist entries...
]
```

It can be seen that the report's "type" matches the "Principal Inspection Report" picklist entry's "id".

The matching picklist value could be returned in JavaScript as follows, assuming the report data is an object called "report" and the picklist data is an array called "reportTypes":

```
reportTypes.filter(reportType => reportType.id === report.type)[0].value
```

4 Exploratory Locations Database (ELDB) API

Full Swagger documentation for the Exploratory Locations Database (ELDB) API is included in the accompanying ExploratoryLocationsDatabase.json file.

4.1 Principal entities

The ELDB module contains the following principal entities:

- Geodataset
- Exploratory location

4.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/geodatasets/{id}>
 - return information for a single geodataset where {id} is the geodataset's GUID
- POST <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/geodatasets/{action}>
 - return information for multiple geodatasets where {action} is "query", "summary" or "csv"
 - the payload of these requests supports some additional, optional properties e.g.:
 - "reportId": a single report GUID as a string, to return geodatasets linked to a report
 - "geodatasetIds": an array of geodataset GUIDs, to return one or more specific geodatasets
- GET <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/exploratorylocations/{id}>
 - return information for a single exploratory location where {id} is the exploratory location's GUID
- POST <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/exploratorylocations/{action}>
 - return information for multiple exploratory locations where {action} is "query", "summary" or "csv"
 - the payload of these requests supports some additional, optional properties e.g.:
 - "geodatasetId": a single geodataset GUID as a string, to return exploratory locations for one geodataset
 - "searchGeometry": a WKT polygon string, to return exploratory locations within a geographical area
- GET <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/exploratorylocations/exploratorylocationtypes>
 - return picklist data for all exploratory location types

The ability to download files such as a geodataset's geodatafile or files attached to exploratory locations is covered in the Files API (see section **Error! Reference source not found.**).

4.3 Worked examples

4.3.1 Return geodatasets linked to a report

Example: return geodatasets linked to report ID 29ebaccd-246d-4263-b835-1d8e48841599.

This example assumes that the report's GUID is known. If only the numeric report ID is known, then you will first need to determine the report's GUID using the worked example in section 3.3.1.

Endpoint: POST <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/geodatasets/summary>

Alternatively the "query" endpoint will return similar data and can be used if this better meets your needs.

The payload of this request can be passed simply as follows, assuming you do not want the API to further filter, sort or paginate the results:

```
{
  "reportId": "29ebaccd-246d-4263-b835-1d8e48841599"
}
```

This will return a standard multiple results response:

```
{
  "results": [array of geodataset objects]
  "count": number of geodatasets linked to the report, considering any filter criteria
  "total": number of geodatasets linked to the report, ignoring any filter criteria
}
```

4.3.2 Return exploratory locations for a geodataset

Example: return exploratory locations for geodataset ID 64a406e2-9909-45dd-b469-7e826d85bc3e.

Having followed the previous example, you may now want to obtain a list of the exploratory locations for a geodataset that was returned.

Endpoint: POST <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/exploratorylocations/summary>

Alternatively the “query” endpoint will return similar data and can be used if this better meets your needs.

The payload of this request can be passed simply as follows, assuming you do not want the API to further filter, sort or paginate the results:

```
{
  "geodatasetId": "64a406e2-9909-45dd-b469-7e826d85bc3e"
}
```

This will return a standard multiple results response:

```
{
  "results": [array of exploratory location objects]
  "count": number of exploratory locations for the geodataset, considering any filter criteria
  "total": number of exploratory locations for the geodataset, ignoring any filter criteria
}
```

4.3.3 Return exploratory locations within a geographical area

Example: return exploratory locations located in a rectangular area around the M25 Junction 6. The bounding box coordinates are 534,500E 153,000N; 535,500E 152,500N.

Endpoint: POST <https://api.gdms.assetia.cloud/exploratorylocationsdatabase/exploratorylocations/summary>

Alternatively the “query” endpoint will return similar data and can be used if this better meets your needs.

There are two different ways to pass the geographical location within the request payload, described below. In both cases, the results will be the same and have the same structure as in example 4.3.2, but the “count” and “total” will be limited to those exploratory locations within the geographical area.

4.3.3.1 Location passed as a Well-Known Text (WKT) polygon

The following payload defines the area of interest as a closed polygon in WKT notation, without any additional filtering, sorting or pagination:

```
{
  "searchGeometry": "POLYGON ((534500 153000, 535500 153000, 535500 152500, 534500 152500, 534500 153000))"
}
```

Although the location being used in this example is a simple north-oriented rectangle, this approach has the flexibility that any WKT polygon may be defined. Note that the polygon must be closed so for a rectangle there are five pairs of coordinates and the last coordinate pair (534500 153000) is the same as the first. All coordinates must be to OS grid.

4.3.3.2 Location passed as filter criteria

Because the location of interest is a simple north-oriented rectangle it is also possible to define this within the payload "filter" object as ranges for the easting and northing fields.

For comparison, the most closely equivalent⁹ SQL of below is:

```
WHERE (easting >= 534500 AND easting <= 535500) AND (northing >= 152500 AND northing <= 153000)
```

The request payload, without any sorting or pagination, is as follows:

```
{
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "AND",
        "filterGroups": [],
        "filters": [
          {
            "operator": "LessThan",
            "not": true,
            "field": "easting",
            "value": 534500
          },
          {
            "operator": "GreaterThan",
            "not": true,
            "field": "easting",
            "value": 535500
          }
        ]
      }
    ],
    "condition": "AND",
    "filterGroups": [],
    "filters": [
      {
        "operator": "LessThan",
        "not": true,
        "field": "northing",

```

⁹ This SQL could also be written using "BETWEEN", but this has not been used as there is no directly equivalent operator in the API filter model.

```
        "value": 152500
      },
      {
        "operator": "GreaterThan",
        "not": true,
        "field": "northing",
        "value": 153000
      }
    ]
  },
  "filters": []
}
```

The combination of the “not” and “operator” properties should be noted, e.g. the following is equivalent to “less than or equal to”:

```
"operator": "GreaterThan",
"not": true
```

5 Network Model and Locations API

Full Swagger documentation for the Network Model and Locations API is included in the accompanying Network.json file.

5.1 Principal entities

The Network Model and Locations module contains the following principal entities, as well as functions for snapping geographical locations to the road network models:

- Area
- Network Geometry (OS)
- Network Geometry (IAM-IS)
- Location (for GAD data)

5.2 Principal endpoint structure

Most of the Network Model and Locations modules API endpoints are not intended for third-party usage, as they relate to snapping new and updated locations to the network. The following endpoints are of relevance for data retrieval:

- GET <https://api.gdms.assetia.cloud/network/areas>
 - return metadata for one or more NH Managing Areas
- GET <https://api.gdms.assetia.cloud/network/locations/{id}>
 - return a single location record where {id} is the location's GUID, as referred to in the GAD record the location is for

5.3 Worked examples

5.3.1 Return the list of Areas including IDs and numbers

Example: return an index of the NH Managing Areas defined within GDMS, including their GUID for linking to other records and Area number.

Endpoint: GET <https://api.gdms.assetia.cloud/network/areas>

The above endpoint is recommended to download all Area information, which should then be cached by your application for the session as it changes very infrequently. Other Area-related endpoints exist to return a single Area or query using standard criteria, if required.

The response includes a “results” array of Area objects, including:

```
{
  "id": GUID of Area for linking to records such as assets and geotechnical events,
  "name": name of Area (e.g. "Area 3"),
  "code": Area code matching the IAM-IS Maintenance Sections Areas (e.g. "A03"),
  "number": number of the Area (e.g. 3),
  "type": type of Area, either "Area", "DBFO" or "Testing",
  "region": name of NH Region the Area is in, e.g. "South East"
}
```

At present, GDMS's Areas are not merged into regions, e.g. Areas 1 and 2 are separate, and their “region” property is “South West”. This may change in future depending on how Areas are defined within updates to the road network model.

5.3.2 Return locations of GAD records

Example: return start and end locations of geotechnical asset 123

This is a two-step process, which will be similar for GAD assets, slope geometries, items and sets:

1. request the asset record of interest, which includes the IDs of its start and end locations
2. using the location IDs, request the detailed location data

Some location data (e.g. chainage) may be included with the GAD record retrieved by step 1, depending on whether you use the /query or /summary endpoints, or retrieve the GAD record by its GUID. Methods to retrieve GAD records are covered in section 6. The full location data is retrieved by step 2.

The below assumes you know the asset number is 123 but not its GUID. Therefore use either the /query or /summary endpoint to search for assets with that number.

Endpoint: POST <https://api.gdms.assetia.cloud/geotechnical/assets/summary>

Payload:

```
{
  "filter": {
    "filters": [
      {
        "operator": "Equals",
        "not": false,
        "field": "visibleId",
        "value": 123
      }
    ]
  }
}
```

If the asset exists this should return a “results” array containing a single asset, which includes “startLocation” and “endLocation” GUIDs. Items also include a “midLocation” GUID if the item was located as mid-point and length, and for slope geometries there is a single “location” GUID.

```
{
  "results": [
    {
      "id": "38ee9fb1-65bf-44f5-8642-8ee3ee378038",
      "visibleId": 123,
      "startLocation": "1ddc7eba-85b6-4d84-974c-8f1ce04ceab3",
      "endLocation": "fcde4b50-6585-402c-9ddb-8d1488bab888",
      other properties omitted...
    }
  ],
  "count": 1,
  "total": 49187
}
```

The start and end location data can then be retrieved as follows:

Endpoint: GET <https://api.gdms.assetia.cloud/network/locations/1ddc7eba-85b6-4d84-974c-8f1ce04ceab3>

Endpoint: GET <https://api.gdms.assetia.cloud/network/locations/fcde4b50-6585-402c-9ddb-8d1488bab888>

6 Geotechnical Assets (GAD) API

Full Swagger documentation for the Geotechnical Assets (GAD) API is included in the accompanying GeotechnicalAssets.json file.

6.1 Principal entities

The GAD module contains the following principal entities:

- Geotechnical asset
- Inventory item (and sub-entities)
- Inventory set
- Condition item (and sub-entities)
- Condition set
- Slope geometry
- Activity (and sub-entities)
- Links between Activities and other entities

Activities, inventory items and condition items each have a set of common fields including a “type”. The type defines other fields for the record, which are stored as a 1:1 linked sub-entity.

6.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/geotechnical/{entity}/{id}>
 - return information for a single GAD record where {entity} is the type of record¹⁰ (e.g. an asset, item, set, slope geometry) and {id} is the record’s GUID
- POST <https://api.gdms.assetia.cloud/geotechnical/{entity}/{action}>
 - return information for multiple GAD records where {entity} is the type of record¹⁰ and {action} is “query”, “search” or “csv”
 - the payload of these requests supports some additional, optional properties e.g.:
 - “includeArchived”: a boolean field to indicate if archived records should be included
 - “searchGeometry”: a WKT polygon string, to return GAD records within a geographical area
- POST <https://api.gdms.assetia.cloud/geotechnical/{setType}/{id}/{itemType}/query>
 - return items related to a set where {setType} is either conditionsets or inventorysets, {id} is the set’s GUID and {itemType} is the corresponding type of item for the type of set
- GET <https://api.gdms.assetia.cloud/geotechnical/picklist...>
 - return picklist related data (see worked example 6.3.1)

The ability to download attached files is covered in the Files API (see section **Error! Reference source not found.**).

¹⁰ please note the exact spelling and pluralisation of the entity in the Swagger JSON documentation

6.3 Worked examples

6.3.1 Picklist data

All GDMS databases use picklist related data, but the GAD database has by far the highest number of picklist-type fields. Some further general information on picklist fields is given in section 2.5.

The GAD API includes endpoints to retrieve:

- all picklist entries for all picklist groups (e.g. for caching within your application):
 - GET <https://api.gdms.assetia.cloud/geotechnical/picklists>
- all picklist group definitions:
 - GET <https://api.gdms.assetia.cloud/geotechnical/picklistgroups>
- a single picklist entry or picklist group when the GUID is already known
 - as above with `/id` appended
- all entries for a specific picklist (e.g. asset types)
 - GET <https://api.gdms.assetia.cloud/geotechnical/{picklistName}>
 - e.g. GET <https://api.gdms.assetia.cloud/geotechnical/assettypes>

These endpoints provide flexibility to either retrieve all picklist related data once, or to request it on demand. GAD picklist data cannot be changed by users and is rarely changed by system administrators, so can be safely cached for a single application session.

6.3.2 Return data for a GAD record where numeric ID only is known

Example: return data for condition set 566259

When returning data for GAD records the following options are available:

1. if the GUID of the record is known, then use an endpoint in the form:
 - GET <https://api.gdms.assetia.cloud/geotechnical/{entity}/{id}>
2. when the GUID is not known, or multiple records meeting particular criteria are required, then use an endpoint in the form of one of the following:
 - POST <https://api.gdms.assetia.cloud/geotechnical/{entity}/summary>
 - POST <https://api.gdms.assetia.cloud/geotechnical/{entity}/query>

The rest of this example will follow (2) above. General information about requesting data for multiple records is given in section 2.6.

The “summary” endpoints in the GAD API return data similar to the “Summary” grids accessed from GDMS’s main menu. These include human readable values for picklist fields, and some additional fields derived from related GAD records (e.g. lengths, maximum slope angles). However, they do not necessarily include all of the fields for the record concerned, and do not include the map location.

The “query” endpoints return raw data for the records and include all fields. However, picklist fields are returned as GUIDs and will need to be converted to readable values, and there will be no supplementary fields derived from other records. The “query” endpoints also include the map location of the GAD record as Well-Known Text (centre-line geometry and offset as displayed).

To return the raw data for condition set 566259:

Endpoint <https://api.gdms.assetia.cloud/geotechnical/conditionsets/query>

Payload:

```
{
  "filter": {
    "filters": [
      {
        "operator": "Equals",
        "not": false,
        "field": "visibleId",
        "value": 566259
      }
    ]
  }
}
```

Assuming condition set 566259 exists, a single result will be returned. This will include the GUID of the record in the “id” property, e.g. “5f72e672-bf9b-46d9-aa9d-b84d8947a907”.

The above only returns the details of the record. See the following examples to retrieve further information:

- if any fields are not included you can use the GUID with the “GET” endpoint above to retrieve the full record
- section 5.3.2 to obtain detailed location data using the location GUIDs
- section 6.3.1 to convert picklist GUIDs into readable values
- section 6.3.3 to return the condition items in this condition set
- section **Error! Reference source not found.** to obtain files attached to the record

6.3.3 Return items within a set

Example: return the condition items within condition set 566259

There are a few ways this can be achieved. The below is how the items are retrieved by the GDMS website, when viewing the “Condition Items” section of a “Condition Set” page. A similar method applies for inventory sets and inventory items.

Worked example 6.3.2 should first be followed if the GUID of the condition set is not known, as this is required in order to return the condition items.

Inventory items and condition items have a set of fields that are common to all inventory item types or condition item types respectively, and then an additional set of fields specific to the type. This example will cover obtaining the common fields only; worked example 6.3.4 will cover requesting the type-specific data.

Endpoint: POST <https://api.gdms.assetia.cloud/geotechnical/conditionsets/5f72e672-bf9b-46d9-aa9d-b84d8947a907/conditionitems/query>

Payload:

```
{}
```

As the endpoint URL includes the set’s GUID, only a limited amount of data will be returned. The payload must be present as a minimum as above, but only needs to include any criteria if you require it to.

This will return a “results” array containing details of all of the condition items currently in the set. As GDMS uses this data in a grid, the returned fields have picklists as readable text, and some additional fields derived from related records are also included.

If further information is required about any of the returned results, then the included “id” may be passed to the condition items endpoint:

Endpoint: GET <https://api.gdms.assetia.cloud/geotechnical/conditionitems/{id}>

6.3.4 Return item type specific fields

Inventory items and condition items have a set of fields that are common to all inventory item types or condition item types respectively, and then an additional set of fields specific to the type. This worked example covers the type-specific fields. The same item GUID is used for the records containing the common and specific fields.

Prior to using the type-specific endpoints, you must request the general item data to determine the type of item. The type-specific endpoints normally use the plural form of the item type.

The GAD API has separate endpoints for each item type, e.g. for “slip” condition items:

- GET <https://api.gdms.assetia.cloud/geotechnical/slips/{id}>
 - obtains a single slip condition item record, where {id} is the GUID of the condition item
 - if the GUID does not match a condition item that is of type “slip” then a 404 response is returned
- POST <https://api.gdms.assetia.cloud/geotechnical/slips/query>
 - obtains one or more slip condition items matching criteria passed in the payload
 - like other “query” endpoints, if the criteria do not match any valid records then no results are returned as opposed to a 404 response

The latter approach supports the standard criteria for requesting multiple records, such that it is possible to use numeric item IDs in the payload “filter” criteria. However, as you will need to determine the item type first, it is more likely you will already know the item GUID. Alternatively, you can specify one or more item GUIDs as follows (this is for condition items):

```
{
  "conditionItemsIds": [
    "d47ff527-72f9-4517-aa31-17ca4935cbe0",
    "52386955-2cfe-4823-89f6-339c446c77fc"
  ]
}
```

If requesting inventory item type-specific data (e.g. vegetation) then instead use “inventoryItemsIds”.

6.3.5 Return assets for negative criteria

Example: return assets in Areas 1 and 2 that are not of type “at grade”

Most of the worked examples in this document use positive criteria. Sometimes it is better to use negative criteria to request data that doesn't match a single value, rather than all the other values that it could. This is because if an additional value was introduced to GDMS in future, your request should still be designed to return all values except the single unwanted value.

The “query” or “summary” endpoint can be used. This example uses the “summary” endpoint so that readable values are used within the request and the response. The process for the “query” endpoint is the same except GUIDs will be used instead for Area and asset type criteria.

Endpoint: POST <https://api.gdms.assetia.cloud/geotechnical/assets/summary>

Payload:

```
{
  "offset": 0,
  "limit": 25,
```

```

"filter": {
  "condition": "AND",
  "filterGroups": [
    {
      "condition": "OR",
      "filterGroups": [],
      "filters": [
        {
          "operator": "Equals",
          "not": false,
          "field": "number",
          "value": 1
        },
        {
          "operator": "Equals",
          "not": false,
          "field": "number",
          "value": 2
        }
      ]
    },
    {
      "condition": "OR",
      "filterGroups": [],
      "filters": [
        {
          "operator": "Equals",
          "not": true,
          "field": "type",
          "value": "At grade"
        }
      ]
    }
  ],
  "filters": []
},
"sort": null,
"includeArchived": false,
}

```

The above “filter” object is equivalent to the following SQL:

WHERE (number = 1 OR number = 2) AND type != 'At grade'

Note in particular how “operator” is affected by the “not” property.

6.3.6 Returning archived GAD data

In GAD requests for multiple records an additional boolean property “includeArchived” may be included in the request payload. This is shown in the worked example 6.3.5.

- When “false” or if not specified, archived records will not be included.
- When “true”, archived records will also be returned if any meet your criteria, and counted in the “count” and “total” within the response.

When requesting an individual record by its GUID, it will be returned regardless of whether it is archived.

Data may also be “deleted” in GDMS. Although in most cases this data is retained in the GDMS databases, such data is not retrievable via the APIs.

7 Geotechnical Events API

Full Swagger documentation for the Geotechnical Events API is included in the accompanying GeotechnicalEvents.json file.

7.1 Principal entities

The Geotechnical Events module contains the following principal entity:

- Geotechnical Event

7.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/events/{id}>
 - return information for a single event where {id} is the event's GUID
- POST <https://api.gdms.assetia.cloud/events/{action}>
 - return information for multiple events where {action} is “query”, “summary” or “summary/csv”
- GET <https://api.gdms.assetia.cloud/events/picklists/{picklistGroup}>
 - return the permitted values and IDs for a picklist field, where {picklistGroup} identifies the field, e.g. “statuses”, “recordstatuses”
- POST <https://api.gdms.assetia.cloud/events/{id}/{action}>
 - (not supported) actions that change data for an individual event where {id} is the event's GUID

The ability to download attached files is covered in the Files API (see section **Error! Reference source not found.**).

7.3 Worked examples

7.3.1 Return geotechnical events by Area and current status

Example: return geotechnical events that are “Ongoing” status in Area 3.

This worked example illustrates the differences between using the “summary” and “query” endpoints, because either could be more suitable to your application. Both will return data in the following structure:

```
{
  "results": [array of geotechnical event objects]
  "count": number of geotechnical events, considering any filter criteria
  "total": number of geotechnical events, ignoring any filter criteria
}
```

7.3.1.1 Using the “summary” endpoint (human-readable data)

The “summary” endpoint returns a smaller number of fields (similar to those in the “Events Summary” grid in GDMS) but requests are sent and returned using human-readable values rather than GUIDs for picklist-type fields.

Endpoint: POST <https://api.gdms.assetia.cloud/events/summary>

Payload (assuming no sorting or pagination):

```
{
  "filter": {
    "condition": "AND",
```

```

"filterGroups": [
  {
    "condition": "OR",
    "filterGroups": [],
    "filters": [
      {
        "operator": "Equals",
        "not": false,
        "field": "status",
        "value": "Ongoing"
      }
    ]
  },
  {
    "condition": "OR",
    "filterGroups": [],
    "filters": [
      {
        "operator": "Equals",
        "not": false,
        "field": "number",
        "value": 3
      }
    ]
  }
],
"filters": []
},
}

```

The above “filter” object is equivalent to the following SQL:

```
WHERE status = 'Ongoing' AND number = 3
```

Note that in the “summary” endpoint’s payload the Area field is called “number”. Correct field naming can be confirmed by testing similar filters using the GDMS system and viewing the API requests in your web browser’s developer tools window.

7.3.1.2 Using the “query” endpoint (raw data)

The “query” endpoint returns all fields for an event, but requests are sent and returned using GUIDs rather than human-readable values for picklist-type fields.

The payload of the request will need to include GUIDs for the required Area and status.

The GUID of Area 3 can be determined by following the worked example in section 5.3.1 and is “4683d605-f68c-4ebd-a1c2-16ad37a602ce”.

The GUID of the “Ongoing” status can be determined as follows.

Endpoint: GET <https://api.gdms.assetia.cloud/events/picklists/statuses>

Returns:

```

[
  {
    "id": "dfcc599e-acd1-4cb0-a88e-18906dd7f96d",

```



```

    "picklistGroup": "c7ea2fff-30af-4708-9aea-0312fc2bfffac",
    "value": "Cleared",
    "description": ""
  },
  {
    "id": "5e1adbd0-0433-4a88-af59-d82aa970469f",
    "picklistGroup": "c7ea2fff-30af-4708-9aea-0312fc2bfffac",
    "value": "Imminent",
    "description": ""
  },
  {
    "id": "d10e37fb-60bf-4a36-b542-68ac5af76f9b",
    "picklistGroup": "c7ea2fff-30af-4708-9aea-0312fc2bfffac",
    "value": "Ongoing",
    "description": ""
  }
]

```

The “Ongoing” picklist value has an “id” of “d10e37fb-60bf-4a36-b542-68ac5af76f9b”.

This could be determined in JavaScript as follows, assuming the above picklist data is an array called “eventStatuses”:

```
eventStatuses.filter(eventStatus => eventStatus.value === "Ongoing")[0].id
```

The request for the event data can now be made.

Endpoint: POST <https://api.gdms.assetia.cloud/events/query>

Payload (assuming no sorting or pagination):

```

{
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "OR",
        "filterGroups": [],
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "status",
            "value": "d10e37fb-60bf-4a36-b542-68ac5af76f9b"
          }
        ]
      }
    ],
  },
  {
    "condition": "OR",
    "filterGroups": [],
    "filters": [
      {
        "operator": "Equals",
        "not": false,
        "field": "area",
        "value": "4683d605-f68c-4ebd-a1c2-16ad37a602ce"
      }
    ]
  }
}

```

```
    }  
  ]  
}  
],  
  "filters": []  
},  
}
```

8 Drainage Catchment Model API

Full Swagger documentation for the Drainage Catchment Model API is included in the accompanying `Catchments.json` file.

In the examples in this section, where an output would show a total number of records this is shown as “###”.

8.1 Principal entities

The Drainage Catchment Model module contains the following entities:

- Catchment
- Sub-catchment

8.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/catchments/{id}>
 - return information for a single catchment where {id} is the GUID of the catchment
- POST <https://api.gdms.assetia.cloud/catchments/summary/{action}>
 - return information for multiple catchments where {action} is “query”, “history” or “filters”
- POST <https://api.gdms.assetia.cloud/catchments/subcatchment/{action}>
 - return information for multiple sub-catchments where {action} is “snap”, “history/query” or “summary/query”,
- POST <https://api.gdms.assetia.cloud/catchments/query>
 - return high-level catchment and sub-catchment data
 - returns data with parameters: id, parentCatchmentId, area, geometry, visibleId, archived

8.3 Worked examples

8.3.1 Determine the GUIDs of all catchments in one Area

Example: determine the GUIDs of all catchments in Area 4.

Firstly, the GUID for Area 4 is required. This can be obtained via the Network module (Section 5):

Endpoint: POST <https://api.gdms.assetia.cloud/network/areas/query>

Payload:

```
{
  "filter": {
    "filters": [
      {
        "field": "number",
        "not": false,
        "operator": "Equals",
        "value": 4
      }
    ]
  }
}
```

This returns (shortened):

```
{
  "results": [
    {
      "id": "71584ef5-4823-4512-8b7c-4b4e6c42e29e",
      "name": "Area 4",
      "number": 4,
      other Area properties...
    }
  ],
  "count": 1,
  "total": 1
}
```

The GUID for Area 4 can be retrieved by `results[0].id`.

Given that we only require the GUIDs of the catchments, rather than all data for each catchment, the endpoint that returns only high-level information can be used. This avoids large amounts of unnecessary data being returned by the API.

Endpoint: POST <https://api.gdms.assetia.cloud/catchments/query>

Payload:

```
{
  "filter": {
    "filters": [
      {
        "field": "area",
        "not": false,
        "operator": "Equals",
        "value": "71584ef5-4823-4512-8b7c-4b4e6c42e29e"
      }
    ]
  }
}
```

The GUID for Area 4 is used as a filter and the endpoint returns (shortened):

```
{
  "results": [
    {
      "id": "bedafd29-b2a8-f08f-131f-6543216261bf",
      "parentCatchmentId": null,
      "area": "71584ef5-4823-4512-8b7c-4b4e6c42e29e",
      "visibleId": "C01234",
      other catchment properties...
    },
    {
      "id": "f04bf934-5d8d-bb0a-8b91-654321dc2010",
      "parentCatchmentId": null,
      "area": "71584ef5-4823-4512-8b7c-4b4e6c42e29e",
      "visibleId": "C01357",
      other catchment properties...
    }
  ]
}
```

```

    },
    other_catchments...
  ],
  "count": ##,
  "total": ###
}

```

The GUIDs returned in the results array can be extracted into a list called `catchmentIdList` in JavaScript by `catchmentIdList = results.map(catchment => catchment.id).`

8.3.2 Using the “subcatchment/snap” endpoint

This endpoint returns details of the nearest sub-catchment(s) to a given point location, and the location of and distance to the nearest point on each of those sub-catchments.

Endpoint: POST <https://api.gdms.assetia.cloud/catchments/subcatchment/snap>

This endpoint accepts a payload made up of the following parameters:

```

{
  "easting": x-coordinate on OS grid,
  "northing": y-coordinate on OS grid,
  "distance": maximum search distance in metres, if unspecified then the distance is capped to 10,000 metres
  "limit": the maximum number of records to return, if unspecified at most only one record will be returned
  "catchmentIds": comma separated list of sub-catchment visible IDs e.g. "SC01234_01,SC01234_02" (optional, to snap to one or more specific sub-catchments regardless of whether there are other, nearer sub-catchments)
}

```

The endpoint imposes an absolute maximum distance of 10km from the location, as it is not intended to be used at long distances from the catchment network or to attempt to return the distance from a single point to a large number of catchments. In GDMS, most records cannot be added at distances further than 200m or 500m depending on the type of record.

Snapping is based on the shortest line between the location and the linear sub-catchment. If the sub-catchment is not adjacent to the location (i.e. there is no possible perpendicular line between the sub-catchment and the location), the nearest point will be at the nearest end of the sub-catchment.

The following examples in this section show the various ways this endpoint can be used to achieve different tasks. In all cases the results are returned in order of distance, nearest first.

8.3.2.1 Return sub-catchments adjacent to specified location

Example: return the nearest 3 sub-catchments to the coordinate pair 390,000E 259,000N.

For this worked example, the payload is as follows:

```

{
  "easting": 390000,
  "northing": 259000,
  "limit": 3
}

```

This will return an array of length 3 containing sub-catchment data as follows (shortened):

```
[
  {
    "catchmentId": "3246c14e-b373-6ed6-6e45-65432111dc71",
    "visibleId": "SC01234_04",
    "parentCatchmentId": "12345642-84c0-70e3-ccfc-301a80c451a8",
    "distance": 392.1193956321281,
    "easting": "390352.55168706778",
    "northing": "258828.34648743307",
    "geometryText": "WKT geometry string...",
    other sub-catchment properties...
  },
  {
    "catchmentId": "1cbee9a8-5c87-71d7-7e4f-654321ca9111",
    "visibleId": "SC01234_03",
    "parentCatchmentId": "12345642-84c0-70e3-ccfc-301a80c451a8",
    "distance": 415.164368,
    other sub-catchment properties...
  },
  {
    "catchmentId": "4f22df2b-466f-48e7-be65-6543213c53a6",
    "visibleId": "SC01234_01",
    "parentCatchmentId": "123456c0-5bfd-eccd-9729-927bcf25a7b2",
    "distance": 706.869549,
    other sub-catchment properties...
  }
]
```

The returned data includes the “distance” in metres to the nearest point on the sub-catchment, and the “easting” and “northing” of that nearest point. The “geometryText” property is the WKT line string representing the sub-catchment, so this can be used to determine if the nearest point coincides with either end of the sub-catchment if you have reason to detect such cases.

If the payload had instead been provided as follows with a “catchmentIds” property, e.g.:

```
{
  "easting": 390000,
  "northing": 259000,
  "limit": 1,
  "catchmentIds": "SC01234_01"
}
```

then only the specified sub-catchment ID would be returned, even though the example above shows it is not the nearest to the location.

If specifying “catchmentIds” then “limit” must be set to a sufficient value to return all of those specified if you require distances to each. If “limit” is omitted, then only the nearest of the specified sub-catchments is returned. The order you specify the “catchmentIds” makes no difference to the returned results, with them always being returned in distance order up to the “limit”. Normally “distance” would not be specified in this case, although the absolute maximum of 10km still applies to the returned data.

8.3.2.2 Determine nearest sub-catchments within 500m of a specified location

Example: Find the nearest 5 sub-catchments within 500m of the coordinate pair 205,025N, 462,160E.

Payload:

```
{  
  "easting": 462160,  
  "northing": 205025,  
  "distance": 500,  
  "limit": 5  
}
```

If there are 5 sub-catchments within the specified distance, an array of length 5 will be returned, otherwise any sub-catchments within the 500m distance will be.

9 Drainage Assets API

Full Swagger documentation for the Drainage Assets API is included in the accompanying DrainageAssets.json file.

9.1 Principal entities

The Drainage Assets module contains the following principal entities:

- Continuous asset
- Point asset
- Region asset
- Activity
- Activity set
- Asset set
- Component
- Observation

9.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/drainageassets/{entity}/{id}>
 - return information for a single entity where {entity} is “continuousasset”, “pointasset”, “regionasset”, “activityset”, “activity”, “assetset”, “component” or “observation” and {id} is the GUID of the entity
- POST <https://api.gdms.assetia.cloud/drainageassets/{entity}/csv>
 - download drainage asset data in csv format, where {entity} is “continuousasset”, “pointasset”, “regionasset”, “activityset”, “activity”, “assetset”, “component” or “observation”
- POST <https://api.gdms.assetia.cloud/drainageassets/{entity}/query>
 - return high-level information for multiple entities of the same type, where {entity} is “continuousasset”, “pointassets”, “regionasset”, “activityset”, “activity”, “assetset”, “component” or “observation”
 - lookup fields are returned as lookup GUIDs rather than readable values
- POST <https://api.gdms.assetia.cloud/drainageassets/{entity}/summary/query>
 - return information for multiple drainage assets, where {entity} is “continuousasset”, “pointassets” or “regionassets”
 - lookup fields are returned as readable text strings, e.g. a code and its definition
- POST <https://api.gdms.assetia.cloud/drainageassets/{entity}summary/query>
 - return information for multiple entities where {entity} is “activityset”, “activity”, “assetset”, “component” or “observation”
 - lookup fields are returned as readable text strings, e.g. a code and its definition
 - if {entity} is “activityset”, there are further endpoints: activitysetsummary/subcatchment/query and activitysetsummary/catchment/query
- POST <https://api.gdms.assetia.cloud/drainageassets/assetActivityHistory/query>
 - return high level information for multiple activities on an asset

9.3 Worked examples

9.3.1 Return asset data for one drainage asset where the Unique Asset Reference only is known

Example: return asset data for a continuous asset with unique asset reference "SX9288_6327c.1".

As the continuous asset's GUID is not known, it is necessary to make a filtered request to a multiple results endpoint. In the Drainage Assets API, you can use a shortened payload when simply filtering to match a specific value in a field – the Swagger API definition files provide the details of all available fields. Use the full filter payload format for more complex criteria (see section 2.6).

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/continuousasset/summary/query>

Payload:

```
{
  "assetRef": "SX9288_6327c.1"
}
```

As there is only one continuous asset with this unique asset reference, the endpoint will return a results array containing one continuous asset's data. This endpoint returns (shortened):

```
{
  "results": [
    {
      "id": "b9524e5c-a894-b3f1-4cbc-02110a38b29e",
      "assetRef": "SX9288_6327c.1",
      "relatedActivityId": "b250ca40-2838-981f-ae80-c90eb6b8bf66",
      other continuous asset properties...
    }
  ],
  "count": 1,
  "total": 1
}
```

If you already knew the asset's GUID rather than its asset reference, you can request the rest of that asset's data as follows, returning the same results as above.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/continuousasset/summary/query>

Payload:

```
{
  "continuousAssetIds": [
    "b9524e5c-a894-b3f1-4cbc-02110a38b29e"
  ]
}
```

Point and region assets can be retrieved in a similar way using these endpoints:

POST <https://api.gdms.assetia.cloud/drainageassets/pointassets/summary/query>

POST <https://api.gdms.assetia.cloud/drainageassets/regionassets/summary/query>

9.3.2 Return observation data for one drainage asset where asset data is known

Example: return observation data for the latest activity for a continuous asset with unique asset reference "SX9288_6327c.1".

Example 9.3.1 must be followed first to determine the relevant activity ID.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/observationsummary/query>

Payload:

```
{
  "assetId": "b9524e5c-a894-b3f1-4cbc-02110a38b29e",
  "activityId": "b250ca40-2838-981f-ae80-c90eb6b8bf66"
}
```

As an alternative to “assetId” you can use “assetRef” with e.g. “SX9288_6327c.1”. However, you will still need to specify the “activityId”. The same endpoint is also used for point and region assets.

The activity ID used above is of the latest activity relating to the asset (“relatedActivityId”), as given by the response in example 9.3.1. This value can be changed to a different activity ID if you want to return the observations for a historic activity. An example of how historic activity IDs can be obtained for an asset is shown in worked example 9.3.3.

Response (shortened):

```
{
  "results": [
    {
      "observationID": "0a2f9d6a-3e55-bee1-f935-e3c1239be9a1",
      "codeType": "inspection",
      other observation properties...
    },
    {
      "observationID": "57d184b0-beac-2e18-01e7-401e96c1a0c2",
      "codeType": "inventory",
      other observation properties...
    },
    {
      "observationID": "803e10aa-1ce9-cdaa-7cd4-2a47f3dd0cec",
      "codeType": "inventory",
      other observation properties...
    }
  ],
  "count": 3,
  "total": 3
}
```

9.3.3 Return activity history for one drainage asset with a known GUID

Example: return the activity history for a point asset with GUID “a8b14f37-cb8d-e8a8-7d73-0000c7359680”.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/assetActivityHistory/query>

Payload:

```
{
  "assetId": "a8b14f37-cb8d-e8a8-7d73-0000c7359680"
}
```

Returns (shortened):

```
{
  "results": [
    other activities...,
    {
      "assetId": "a8b14f37-cb8d-e8a8-7d73-0000c7359680",
      "assetRef": "TL1234_5678f",
      "supplierRef": "XXXX5678",
      "activityId": "33ddf506-b64a-a522-fd83-17ba5269a578",
      "activityDate": "2021-04-28T03:03:00",
      "activityType": "U - Update",
      other activity properties...
    },
    other activities...
  ],
  "count": 3,
  "total": 3
}
```

Note – this only gives basic details of each activity such as the ID, date and type. To get full details the “/activitysummary/query” endpoint is required, with the activity ID(s):

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/activitysummary/query>

Payload:

```
{
  "ActivityIds": [
    "33ddf506-b64a-a522-fd83-17ba5269a578",
    "fe677116-4cd8-f31f-690e-aa9cdaf107fd",
    "ae939b4f-2e37-650a-1e14-d3ef65ad8609"
  ]
}
```

9.3.4 Return asset data for a component with a known GUID

Example: return asset data for a component with GUID “004645cd-f253-9b4f-a00e-e1302678e187”.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/componentssummary/query>

Payload:

```
{
  "componentIds": [
    "004645cd-f253-9b4f-a00e-e1302678e187"
  ],
}
```

Returns (shortened):

```
{
  "results": [
    {
      "assetRef": "NT9751_3993d.1",
      "assetType": "PW - Pipework",
      "componentId": "004645cd-f253-9b4f-a00e-e1302678e187",

```

```

        "assetId": "5dc444db-c99e-78b5-5d09-841cfb4edf83",
        "geomType": "Continuous",
        other component properties...
    }
],
"count": 1,
"total": 1
}

```

This endpoint has returned a results array containing one component with an asset ID and a geometry type of continuous. The “continuousasset/summary/query” endpoint can be used to return information on the asset.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/continuousasset/summary/query>

Payload:

```

{
  "ContinuousAssetIDs": [
    "5dc444db-c99e-78b5-5d09-841cfb4edf83"
  ],
}

```

Returns (shortened):

```

{
  "results": [
    {
      "id": "5dc444db-c99e-78b5-5d09-841cfb4edf83",
      "assetRef": "NT9751_3993d.1",
      "supplierRef": "PW3993A.1",
      other continuous asset properties...
    }
  ],
  "count": 1,
  "total": 1
}

```

9.4 Data round tripping

Uploading and downloading data is a relatively complex use of the API, with higher permissions required than all the viewing endpoints described through the rest of this document. The scope of this document is to explain how to use the relevant endpoints to achieve this, but not to cover general information such as the format of the round-tripped data, data acceptance criteria or the user account permissions required, which are no different to using the GDMS website user interface.

Prior to attempting to use the round-tripping endpoints you should familiarise yourself with the following:

- Drainage Data Formats document available from the GDMS Downloads page¹¹
- Training courses for “Drainage asset data”, Manage and Review levels¹²

¹¹ <https://downloads.gdms.assetia.cloud>

¹² <https://www.supplychainschool.co.uk/national-highways-geotechnical-drainage-management-service-gdms/>

9.4.1 Principal endpoint structure

These are the main types of endpoint for drainage asset system round tripping in Shapefile data format:

Downloading data

- POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/export/schedule>
 - create a new download task for the export of drainage asset data for one or more drainage systems
 - returns an ID for the download task
- POST <https://api.gdms.assetia.cloud/drainageassets/job/{id}>
 - return details and preparation status of a download task, where {id} is the download task ID
- GET <https://api.gdms.assetia.cloud/drainageassets/downloaddata/{id}>
 - download a ZIP file of asset data where {id} is the ID of a download task that has completed preparation

Uploading data

- POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/validate/{activitySetReference}/{checkPermission}>
 - create a new upload task, where {activitySetReference} is the Activity Set Reference (string) and {checkPermission} is “true” or “false” depending on whether the upload is for checking only
 - returns an ID for the upload task
- GET <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/details/{id}>
 - return details and status of an upload task, where {id} is the ID of the upload task
- GET <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/importedfile/{id}>
 - return imported file information for an upload task where {id} is the ID of the upload task
- POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/{errorType}>
 - return error information for an upload task where {errorType} is “errors”, “warnings” or “datalosswarnings”
- POST <https://api.gdms.assetia.cloud/drainageassets/job/checkResult/export/csv>
 - return check results from an upload task in CSV format, requires the ID of the upload task
- PATCH <https://api.gdms.assetia.cloud/drainageassets/assetsystem/import/schedule/{id}>
 - schedule the import of an upload task, where {id} is the ID of the upload task
- GET <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/ActivitySetId/{importFileID}>
 - return the activity set ID for an imported file where {importFileID} is the ID of the imported file
- POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/{id}/closetask>
 - close an upload task, where {id} is the ID of the upload task

9.4.2 Downloading Asset Systems

9.4.2.1 For all asset systems in a single catchment

Example: Download all asset systems in catchment “C00580”.

From the Drainage Catchment Model module, the “catchments/summary/query” endpoint (Section 8.2) can be used with a filter on “visibleId” to determine the GUID of the catchment. The GUID is obtained from the results array by `results[0].catchmentId`, which gives a value of **“e879720f-67d3-f2cd-908e-61e240e54c12”**.

With the catchment’s ID known, a request can be made to the “drainageassets/pointassets/summary/query” endpoint (Section 9.2) to get the Area and asset system IDs for all **non-archived** point assets in the

catchment. It is not necessary to similarly query the continuous assets and region assets, as these cannot exist in a system without being connected to point assets.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/pointassets/summary/query>

Payload:

```
{
  "catchmentId": "e879720f-67d3-f2cd-908e-61e240e54c12",
  "archivedFlag": false
}
```

Alternatively, you can directly query the point assets endpoint with the catchment's visible ID ("C00580") as follows, still ensuring that you request only **non-archived assets**:

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/pointassets/summary/query>

Payload:

```
{
  "archivedFlag": false,
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "AND",
        "filterGroups": [],
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "CatchmentVisibleId",
            "value": "C00580"
          }
        ]
      }
    ]
  },
  "filters": []
}
```

By either method, the endpoint returns a results array containing the information for 275 point assets, each with a "systemId" and "area":

```
{
  "results": [
    {
      other point asset properties...,
      "systemId": "2B530676",
      "area": 4,
      other point asset properties...
    },
    {
      other point asset properties...,
      "systemId": "B4DAD99C",
      "area": 4,
      other point asset properties...
    }
  ]
}
```

```

    },
    other point assets...
  ],
  "count": 275,
  "total": 275
}

```

These system IDs and Areas are required to schedule the download task with the “export/schedule” endpoint. To schedule the download of drainage asset data, a download reference is needed. This should be something recognisable to the user and ideally includes the date that the data was downloaded. In this example, the asset systems have not been locked on download (see “locked”: false in the payload below).

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/export/schedule>

Payload:

```

{
  "assetSystems": [
    "2B530676",
    "B4DAD99C",
    other asset system IDs...
  ],
  "locked": false,
  "downloadReference": "C00580_20240910",
  "areas": "4"
}

```

Note the “areas” parameter must be a string containing each of the different Area numbers corresponding to the asset system IDs listed in the “assetSystems” array. In this case all asset systems are in Area 4, as this is the Area of the catchment.

The “locked” parameter can be set to true if you wish to lock the asset systems after downloading them. This would prevent them being downloaded again until they are unlocked.

“downloadReference” can be any string that you can use to visibly identify the task, especially if a user would need to find this task in the GDMS dashboard user interface. You should normally make this string unique to you, and including the date would help with this.

The response is a GUID, e.g. “a8b8cb5d-db0a-4cd2-8fd7-30fe5522f6cd”, which is the job ID of the newly created download task.

The “stage” of the download task can be checked and refreshed using the “job/{id}” endpoint.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/job/a8b8cb5d-db0a-4cd2-8fd7-30fe5522f6cd>

No payload is required.

The initial response is (shortened):

```

{
  other download task properties...,
  "stage": "Download being prepared",
  other download task properties...
}

```

The response a while later is (shortened):

```
{
  other download task properties...,
  "stage": "Download ready",
  other download task properties...
}
```

The amount of time that a download will take to be ready depends on the volume of asset data in the asset system(s), and the number of any other downloads also being prepared for the current user or other users. Although an indication of the data volume may be given by the number of point assets (275 in this example), it is not possible to use the API to view information about other users' download tasks. As a guide, if your code will need to automatically poll the "job/{id}" endpoint for progress updates, then a one-minute interval is likely to be suitable.

Now that the download is ready, the SHP data can be downloaded using the "downloadData/{id}" endpoint, which will provide a ZIP file containing various data files.

Endpoint: GET <https://api.gdms.assetia.cloud/drainageassets/downloadData/a8b8cb5d-db0a-4cd2-8fd7-30fe5522f6cd/>

If you attempt to request the ZIP file before the download is ready, then the above endpoint will return a 500 error. The download will continue to be prepared and you will be able to make another request to download the file once it is ready.

9.4.2.2 For a single asset system

Example: Download all asset data for the drainage system that the continuous asset with GUID "c65bcf86-b9fd-b538-15f1-0012471a46fb" is a part of and do not lock the asset system.

If you do not know the asset's GUID but do know its asset reference, then follow example 9.3.1 to determine its GUID and asset system ID.

The first step is to determine the system ID of the continuous asset.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/continuousasset/summary/query>

Payload:

```
{
  "ContinuousAssetIDs": [
    "c65bcf86-b9fd-b538-15f1-0012471a46fb"
  ]
}
```

Response (shortened):

```
{
  "results": [
    {
      "id": "c65bcf86-b9fd-b538-15f1-0012471a46fb",
      "systemId": "B3F7D7A5",
      "area": 4,
      other continuous asset properties...
    }
  ],
}
```



```

    "count": 1,
    "total": 1
}

```

The download can be scheduled with the “assetsystem/export/schedule” endpoint.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/export/schedule>

Payload:

```

{
  "assetSystems": [
    "B3F7D7A5"
  ],
  "locked": false,
  "downloadReference": " B3F7D7A5 20240910",
  "areas": "4"
}

```

Response: "3ea4df1c-8cb2-46a6-9831-1dd6e84a8880"

Repeat the steps in worked example 9.4.2.1 for refreshing the job details and downloading the SHP files with the job ID given above.

9.4.3 Uploading Asset Systems

9.4.3.1 Uploading of data requires the data to be in a certain format, an activity set reference, and suitable access permissions. These are described further in the materials referenced at the start of section 9.4 Without errors

Example: Upload a zip file called “GDMS-1996b.zip”, with an activity set reference of “GDMS-1996b 20240910”, where the upload is not for checking only. This is a test file that is known to not have any errors.

To create the upload task, the “upload/validate” endpoint is used, the URL of which is created from the template below.

<https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/validate/{activitySetReference}/{checkPermission}>

Note that any spaces in the activity set reference will need to be replaced with a “%” symbol in the URL, as intentionally included in this example; however, an underscore or hyphen is recommended instead of a space. Because the upload task is not for checking only, the {checkPermission} part of the endpoint is set to false.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/validate/GDMS-1996b%20240910/false>

The contents of the .zip file needs to be sent in the POST request to the endpoint. To achieve this in Python using the requests package, assuming the “headers” dictionary has been set up with the access token (section 2.1.2) and “url” is the endpoint shown above:

```

files = {"package": ("GDMS-1996b.zip", open(file_path, "rb"), "application/zip")}
response = requests.post(url=url, headers=headers, files=files)

```

where “file_path” is the path to the “GDMS-1996b.zip” file.

Given the request is successful, the response is a GUID, e.g. "6c372623-4c84-4ee1-9fc9-ab5d3a835ece". This is the ID of the newly created upload task. which will be used to check the progress of the task and to schedule the upload.

Next, the details of the upload task need to be checked repeatedly until the Task stage is no longer "Check in progress". To do this, the "upload/details" endpoint is used with the upload task ID returned from the "upload/validate" endpoint.

Endpoint: GET <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/details/6c372623-4c84-4ee1-9fc9-ab5d3a835ece>

The response from the API immediately after the upload task has been created is as follows (shortened):

```
{
  "jobId": "6c372623-4c84-4ee1-9fc9-ab5d3a835ece",
  other upload task properties...,
  "taskStage": "Check in progress",
  other upload task properties...
}
```

Requests can be sent until the response is (shortened):

```
{
  "jobId": "6c372623-4c84-4ee1-9fc9-ab5d3a835ece",
  other upload task properties...,
  "taskStage": "Check successful",
  other upload task properties...,
  "criticalErrors": 0,
  "errors": 0,
  "dataLossWarnings": 0,
  "otherWarnings": 6
}
```

Note that the "taskStage" has now updated to "Check successful" and the number of critical errors and errors is zero.

Now that the checks have been completed successfully, the upload task can be scheduled using the "import/schedule" endpoint.

Endpoint: PATCH <https://api.gdms.assetia.cloud/drainageassets/assetsystem/import/schedule/6c372623-4c84-4ee1-9fc9-ab5d3a835ece>

The import is scheduled by providing a title and changing the Set type to "data upload". Note, by doing this you confirm that you wish to schedule the import of the dataset and acknowledge that this action is irreversible.

Using a title of "Example title", the payload is as follows:

```
[
  {
    "op": "replace",
    "path": "/title",
    "value": "Example title"
  },
  {
    "op": "replace",
    "path": "/setType",
    "value": "data upload"
  }
]
```

```

    }
  ]

```

The successful response of this request is (shortened):

```

{
  "importFileID": "e3db1fa5-5f8d-4efd-b908-31ca34d3a8e4",
  "jobID": "6c372623-4c84-4ee1-9fc9-ab5d3a835ece",
  "importedBy": "4145b1e1-d1b1-427a-9fc1-f63bae24fef5",
  "fileName": "GDMS-1996b.zip",
  other schedule import properties...,
  "setType": "data upload",
  "title": "Example title",
  other schedule import properties...
}

```

A 400 error will be returned instead if there is a reason that the task cannot be imported, for example errors were found during checking (see example 9.4.3.2), it has already been imported or another task has been imported in parallel that now invalidates this task's check results. Some time-critical scenarios such as the latter are not possible to detect until the import scheduling is attempted. Such a failure is permanent for a task and, if the data still needs to be imported, it would need to be re-uploaded as a new task.

Assuming no error was returned, returning to the details of the upload task, the Task stage will now show as "Import in progress".

Endpoint: GET <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/details/6c372623-4c84-4ee1-9fc9-ab5d3a835ece>

Response:

```

{
  "jobId": "6c372623-4c84-4ee1-9fc9-ab5d3a835ece",
  "importFileID": "e3db1fa5-5f8d-4efd-b908-31ca34d3a8e4",
  other upload task properties...,
  "taskStage": "Import in progress",
  other upload task properties...
}

```

This can be refreshed until the Task stage changes to "Import completed". Once the import is complete, no further action is required but, if needed, the activity set ID created during the import can be retrieved using the "upload/ActivitySetId" endpoint and the "importFileID" given above.

Endpoint: GET <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/ActivitySetId/e3db1fa5-5f8d-4efd-b908-31ca34d3a8e4>

Response:

```
"233c2f9a-6947-455b-8534-5373ac12ad8a"
```

To view the details of the activity set, the "activitysetsummary/query" endpoint from Section 9.2 can be used.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/activitysetsummary/query>

Payload:

```
{
```

```

    "ActivitySetIds": [
      "233c2f9a-6947-455b-8534-5373ac12ad8a"
    ],
    "ArchivedFlag": true
  }
}
Response (shortened):
{
  "results": [
    {
      "activitySetId": "233c2f9a-6947-455b-8534-5373ac12ad8a",
      "visibleId": 25392,
      "area": "50",
      "areaID": "6CE5C5F1-02A3-48DF-87CF-BED517C05395",
      "setReference": "GDMS-1996b 20240910",
      "title": "Example title",
      other activity set properties...,
      "jobID": "6c372623-4c84-4ee1-9fc9-ab5d3a835ece",
      "importFileID": "e3db1fa5-5f8d-4efd-b908-31ca34d3a8e4"
    }
  ],
  "count": 1,
  "total": 1
}

```

Note that the activity set details include the “jobID” and “importFileID” for the original upload task, allowing the data associated with the upload task to be found in future, including for example who uploaded it or whether any warnings were identified during the checking.

Using the Activity Set ID, other endpoints in the Drainage API can then be queried to return details of the assets and activities that were imported. For example:

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/activitysummary/query>

Payload:

```

{
  "ActivitySetId": "233c2f9a-6947-455b-8534-5373ac12ad8a",
  "AssetGeometryType": "Point"
}

```

“AssetGeometryType” is optional and can be “Point”, “Continuous” or “Region” to only return assets of that type. If omitted, then all imported assets are returned. Only limited fields are returned, that are common to all three asset geometry types. Returned fields include “assetGeometryType”, “assetId” and “assetRef”, which will allow you to query specific asset endpoints for full asset details (see example 9.3.1).

9.4.3.2 With errors

Example: Upload a zip file called “GDMS-1996a.zip”, with an activity set reference of “GDMS-1996a 20240910”, where the upload is not for checking only. This is a test file that is known to contain an error.

As with worked example 9.4.3.1, the upload task is first validated.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/validate/GDMS-1996a%2020240910/false>

A POST request is sent to the above endpoint with a payload containing the .zip file. The response is:

"f1fcb360-46ad-4555-938f-70666042c939"

The details of the upload task are refreshed until the check is complete using the "upload/details" endpoint.

Endpoint: GET <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/details/f1fcb360-46ad-4555-938f-70666042c939>

Response (shortened):

```
{
  other upload task details...,
  "taskStage": "Check in progress",
  other upload task details...
}
```

Response (shortened) a while later:

```
{
  other upload task details...,
  "taskStage": "Check completed with error(s)",
  other upload task details...,
  "criticalErrors": 0,
  "errors": 1,
  "dataLossWarnings": 0,
  "otherWarnings": 6
}
```

The details of the upload task show that the Task stage is "Check completed with error(s)" and there is 1 error. The upload task cannot proceed until all errors are addressed.

To return information on the error, the "upload/{errorType}" endpoint is used.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/errors>

Payload:

```
{
  "jobId": "f1fcb360-46ad-4555-938f-70666042c939"
}
```

Response:

```
{
  "results": [
    {
      "jobId": "f1fcb360-46ad-4555-938f-70666042c939",
      "importFileID": "d233e3e8-e4bb-44e7-9689-208a9237943d",
      other error properties...,
      "errorMessage": "Asset could not be linked to a sub-catchment because all of the
assets in the same system are more than 200m from any sub-catchment",
      other error properties...
    },
    any other errors...
  ],
  "count": 1,
  "total": 1
}
```

To download the check results as a CSV, the “job/checkResult/export/csv” is used.

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/job/checkResult/export/csv>

Payload:

```
{
  "jobId": "f1fcb360-46ad-4555-938f-70666042c939"
}
```

Note that attempting to schedule import of an upload task with errors will result in a 400 error being returned by the server.

9.4.3.3 Closing a task

Example: Close the upload task created in worked example 9.4.3.1 with ID "6c372623-4c84-4ee1-9fc9-ab5d3a835ece".

Endpoint: POST <https://api.gdms.assetia.cloud/drainageassets/assetsystem/upload/6c372623-4c84-4ee1-9fc9-ab5d3a835ece/closetask>

If the user has access permissions to close a task, sending a POST request to the endpoint above will give a response of “true” and the upload task will be closed.

Note that the same endpoint is used for closing download tasks.

10 Priority Assets API

Full Swagger documentation for the Priority Assets API is included in the accompanying PriorityAssets.json file.

10.1 Principal entities

The Priority Assets module contains the following principal entities:

- Priority Asset Risk
- Priority Asset Workflow
- Priority Asset Workflow Activity

These entities are provided for each of the three types of priority asset: culvert, outfall and soakaway.

10.2 Principal endpoint structure

These are the main types of endpoint for this API:

- POST <https://api.gdms.assetia.cloud/priorityassets/{asset}/{type}/{action}>
 - return information for multiple priority assets where {asset} is “culvert”, “outfall” or “soakaway”, {type} is “risk” or “workflow” and {action} is “summary”, “history” or “export”
- POST <https://api.gdms.assetia.cloud/priorityassets/{asset}workflow/summary/activity>
 - return information for multiple activities for priority asset workflows where {asset} is “culvert”, “outfall” or “soakaway”
- POST <https://api.gdms.assetia.cloud/priorityassets/workflow/stages>
 - return information for multiple workflow stages, the payload of these requests supports some additional, optional properties, e.g.:
 - "fundingRouteID": a single funding route GUID sent as a string, to return workflow stages for one funding route
 - "workflowStatusID": a single workflow status GUID sent as a string, to return workflow stages for one workflow status
- GET <https://api.gdms.assetia.cloud/priorityassets/{asset}/{type}/summary/{id}>
 - return information for a single priority asset where {asset} is “culvert”, “outfall” or “soakaway”, {type} is “risk” or “workflow”, and {id} is the GUID of the priority asset
- GET <https://api.gdms.assetia.cloud/priorityassets/picklists/{picklistGroup}>
 - return the permitted values and IDs for a picklist field, where {picklistGroup} identifies the field, e.g. “solublepollution”, “mitigationTypes”

10.3 Worked examples

10.3.1 Return risk data for one soakaway asset with a known GUID

Example: return risk data for one soakaway asset with a GUID of “51c51b7a-fcbb-89e7-6c7e-cf6221bf00b6”.

Endpoint: POST <https://api.gdms.assetia.cloud/priorityassets/soakawayrisk/summary/query>

The GUID is used in the payload as follows:

```
{
  "soakawayRiskIds": [
```

```

        "51c51b7a-fcbb-89e7-6c7e-cf6221bf00b6"
    ]
}

```

If the soakaway risk ID is valid, this returns (shortened):

```

{
  "results": [
    {
      "soakawayID": "51c51b7a-fcbb-89e7-6c7e-cf6221bf00b6",
      "assetRef": "NY0123_4567b",
      "geomType": "Point",
      "assetTypeCode": "S0 - Soakaway Chamber",
      "area": 13,
      other soakaway properties...
    }
  ],
  "count": 1,
  "total": 1
}

```

The “soakawayRiskId” and “soakawayID” are the same as each other and the asset’s GUID in the Drainage Assets module.

The approach for culverts and outfalls is the same, replacing “soakaway” with “culvert” or “outfall” throughout.

If you only know the asset reference of the soakaway rather than its GUID, you can either follow worked example 9.3.1 to determine the asset’s GUID and then make the above request, or you can add full filter criteria (see section 2.6.5) against the “assetRef” field to the above request.

10.3.2 Return workflow data for one culvert asset with a known GUID

Example: Return workflow data for one culvert asset with a GUID of “2e54db93-9084-2a8e-b8a6-d39ac310a4e8”.

Endpoint: POST <https://api.gdms.assetia.cloud/priorityassets/culvertworkflow/summary/query>

Payload:

```

{
  "culvertRiskIds": [
    "2e54db93-9084-2a8e-b8a6-d39ac310a4e8"
  ],
}

```

Response:

```

{
  "results": [
    {
      "workflowId": "5ecda017-7c02-c9d9-2a6d-5b7c25500444",
      "workflow": 20247,
      "subCatchment": "SC02227_01",
      "culvertId": "2e54db93-9084-2a8e-b8a6-d39ac310a4e8",
      other workflow details...
    }
  ]
}

```



```

    ],
    "count": 1,
    "total": 1
  }
}

```

The approach for outfalls and soakaways is the same, replacing “culvert” with “outfall” or “soakaway” throughout.

It should be noted that most priority assets do not have any workflows and therefore an empty “results” array will commonly be returned.

10.3.3 Return culvert asset data by workflow status and sub-catchment

Example: return priority asset data for culverts with workflow status “In progress” and within sub-catchment “SC02279_04”, not including archived assets.

Endpoint: POST <https://api.gdms.assetia.cloud/priorityassets/culvertrisk/summary/query>

Payload:

```

{
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "OR",
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "currentWorkflowStatus",
            "value": "In Progress"
          }
        ]
      },
      {
        "condition": "OR",
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "subCatchment",
            "value": "SC02279_04"
          }
        ]
      }
    ]
  },
  "IncludeArchived": false
}

```

The above “filter” object is equivalent to the following SQL:

```
WHERE currentWorkflowStatus = 'In progress' AND subCatchment = 'SC02279_04'
```

This end point returns (shortened):

```
{
  "results": [
    {
      "culvertId": "df19bd6a-da14-3d9b-e3e0-2b62913b7aea",
      "assetRef": "SE1234_5678h.1",
      "assetTypeCode": "CU - Culvert",
      "area": 12,
      other culvert properties...
    },
    {
      "culvertId": "ef012c62-e8bc-f059-96be-bcccd81f7650",
      "assetRef": "8765_4321b.1",
      "assetTypeCode": "CU - Culvert",
      "area": 12,
      other culvert properties...
    }
  ],
  "count": 2,
  "total": 7823
}
```

The approach for outfalls and soakaways is the same, replacing “culvert” with “outfall” or “soakaway” throughout.

Any of the fields returned in the “results” array of the endpoint can be used in the filter criteria. For example, include a criterion on the numeric “area” field to filter on assets within an Area.

10.3.4 Determine activity history for an outfall workflow with a known GUID

Example: Determine the activity IDs related to an outfall workflow with a GUID of “7942364e-1213-b100-ad0b-ee2180eddea2”. To determine the workflow ID you may need to use some of the previous worked examples in this section, depending on the information you already have.

Endpoint: POST <https://api.gdms.assetia.cloud/priorityassets/outfallworkflow/summary/activity>

Payload:

```
{
  "outfallWorkflowIds": [
    "7942364e-1213-b100-ad0b-ee2180eddea2"
  ]
}
```

Response (shortened):

```
{
  "results": [
    {
      "visibleId": 55890,
      "date": "2019-10-28T11:23:17.713",
      "workflow": 55890,
      other activity summary properties...,
      "activityId": "b131c48f-8202-baa5-bb40-73a9a97e7675",
    }
  ]
}
```

```
        "workflowId": "7942364e-1213-b100-ad0b-ee2180eddea2",
        "workflowStage": "Other - Desk study complete - field study required (from migrated
priority asset)"
      }
    ],
    "count": 1,
    "total": 1
  }
}
```

The most useful information returned by this endpoint will be the “date” and “workflowStage”. All workflows should include at least one activity, with the most recent activity’s stage corresponding with the workflow’s current stage.

The approach for culverts and soakaways is the same, replacing “outfall” with “culvert” or “soakaway” throughout.

11 Floods API

Full Swagger documentation for the Floods API is included in the accompanying Floods.json file.

In the examples in this section, where an output would show a total number of records this is shown as “###”.

11.1 Principal entities

The Floods module contains the following principal entity:

- Flood

11.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/floods/flood/{id}>
 - return information for a single flood where {id} is the flood’s GUID
- POST <https://api.gdms.assetia.cloud/floods/flood/{action}>
 - return information for one or multiple floods where {action} is “query”, “summary” or “history”
- GET <https://api.gdms.assetia.cloud/floods/picklists/{picklistGroup}>
 - return the permitted values and IDs for a picklist field, where {picklistGroup} identifies the field, e.g. “reportedbys”, “maximumeffects”
- POST <https://api.gdms.assetia.cloud/floods/csv/floods>
 - return flood data in CSV format

11.3 Worked examples

11.3.1 Return data for a flood record where numeric ID only is known

Example: return data for a flood with numeric ID 123.

As the flood GUID is not known, it is necessary to make a filtered request to a multiple results endpoint.

Endpoint: POST <https://api.gdms.assetia.cloud/floods/flood/summary/query>

As there is only one criterion, the shortened version of the “filter” payload can be used (section 2.6.5.1):

```
{
  "filter": {
    "filters": [
      {
        "field": "visibleId",
        "not": false,
        "operator": "Equals",
        "value": 123
      }
    ]
  }
}
```

As there is only one flood with a numeric ID of 123, provided this flood exists, the endpoint will return a results array containing one flood's data. This endpoint returns data including human-readable picklist values as follows (shortened):

```
{
  "results": [
    {
      "id": "63e1d674-b747-4ee2-8631-db4f123e218c",
      "visibleId": 123,
      "easting": 552753.000000,
      "northing": 344113.000000,
      "geometry": "POINT (552753 344113)",
      "reportedDateTime": "2024-06-03T10:46:00",
      other flood properties...
    }
  ],
  "count": 1,
  "total": ###
}
```

If other data for the flood is needed, then the flood's GUID can be retrieved as `results[0].id` and passed into an appropriate endpoint.

11.3.2 Return data for one flood with a known GUID and determine its maximum effect

Example: return data for a flood with GUID "63e1d674-b747-4ee2-8631-db4f123e218c" and determine its maximum effect.

11.3.2.1 Using the "{id}" and "{picklistGroup}" endpoints

As the flood's GUID is known, the specific flood can be requested.

Endpoint: GET <https://api.gdms.assetia.cloud/floods/flood/63e1d674-b747-4ee2-8631-db4f123e218c>

If the flood exists, it will return data as follows (shortened):

```
{
  "id": "63e1d674-b747-4ee2-8631-db4f123e218c",
  "visibleId": 123,
  "easting": 552753.000000,
  "northing": 344113.000000,
  other flood properties...,
  "maximumEffect": "5619f303-bb62-407b-a434-34ba436eb3cf",
  other flood properties...
}
```

The maximum effect of the flood is given by the GUID in the "maximumEffect" parameter. This needs to be converted to its equivalent human-readable picklist value using the "picklists/maximumeffects" endpoint.

Endpoint: GET <https://api.gdms.assetia.cloud/floods/picklists/maximumeffects>

This will return an array of all maximum effects as follows (shortened):

```
[
  other picklist entries...,
  {
    "id": "5619f303-bb62-407b-a434-34ba436eb3cf",
    "picklistGroup": "e07946a1-3522-4855-9e58-1152080dc8f3",
    "value": "Partial Closure",
    "description": ""
  },
  other picklist entries...
]
```

The flood's "maximumEffect" GUID matches the "Partial Closure" picklist entry's "id".

The matching picklist value could be returned in JavaScript as follows, assuming the flood data is an object called "flood" and the picklist data is an array called "maximumEffects":

```
maximumEffects.filter(maximumEffect => maximumEffect.id === flood.maximumEffect)[0].value
```

11.3.2.2 Using the "summary/query" endpoint (human-readable data)

Alternatively, the maximum effect can be determined within one API request by:

Endpoint: POST <https://api.gdms.assetia.cloud/floods/flood/summary/query>

Payload:

```
{
  "floodIds": [
    "63e1d674-b747-4ee2-8631-db4f123e218c"
  ]
}
```

As there is only one flood with a GUID of "63e1d674-b747-4ee2-8631-db4f123e218c", the endpoint will return a results array containing one flood's data (shortened):

```
{
  "results": [
    {
      "id": "63e1d674-b747-4ee2-8631-db4f123e218c",
      "visibleId": 123,
      "easting": 552753,
      "northing": 344113,
      other flood properties...,
      "maximumEffect": "Partial Closure",
      other flood properties...
    }
  ],
  "count": 1,
  "total": ###
}
```

11.3.3 Return floods by reported date and status

Example: Return floods that have a reported date between 01/01/2015 and 31/12/2015 and a status of "Historic".

Endpoint: POST <https://api.gdms.assetia.cloud/floods/food/summary/query>

Payload:

```
{
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "OR",
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "status",
            "value": "Historic"
          }
        ]
      },
      {
        "condition": "AND",
        "filters": [
          {
            "operator": "LessThan",
            "not": true,
            "field": "reportedDateTime ",
            "value": "2015-01-01"
          },
          {
            "operator": "LessThan",
            "not": false,
            "field": "reportedDateTime ",
            "value": "2016-01-01"
          }
        ]
      }
    ]
  },
}
```

The above "filter" object is equivalent to the following SQL:

```
WHERE status = 'Historic' AND (reportedDateTime >= '2015-01-01' AND reportedDateTime < '2016-01-01')
```

Because the “reportedDateTime” is a date/time field, to include all floods at any time on 31/12/2015 you should query for dates less than 01/01/2016. To query a narrow time band it is possible to also include a time in the criteria, e.g. 10.30am on 01/01/2015 would be “2015-01-01T10:30:00.000Z”.

This endpoint returns data for floods matching the filter criteria (shortened):

```
{
  "results": [
    {
      "id": "105a790e-d096-253b-9125-2c1b0c6adaf2",
      "visibleId": 123,
      other flood properties...
    },
    {
      "id": "f796e096-a8a8-51aa-5d3d-f357da969504",
      "visibleId": 456,
      other flood properties...
    },
    other flood results...
  ],
  "count": ##,
  "total": ###
}
```


12 Flood Risks API

Full Swagger documentation for the Flood Risks API is included in the accompanying FloodRisks.json file.

12.1 Principal entities

The Flood Risks module contains the following principal entities:

- Sub-catchment Flood Risk
- Flood Risk Workflow
- Flood Risk Workflow Activity

12.2 Principal endpoint structures

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/floodrisks/workflow/summary/{id}>
– return information for a single workflow where {id} is GUID of the workflow
- GET <https://api.gdms.assetia.cloud/floodrisks/floodrisk/summary/{id}>
– return flood risk information for a single sub-catchment where {id} is the GUID of the sub-catchment
- POST <https://api.gdms.assetia.cloud/floodrisks/workflow/{action}>
– return information for multiple flood risk workflows where {action} is “summary”, “history” or “csv”
- POST <https://api.gdms.assetia.cloud/floodrisks/floodrisk/{action}>
– return flood risk information for multiple sub-catchments where {action} is “summary”, “history” or “csv”
- POST <https://api.gdms.assetia.cloud/floodrisks/workflow/activity/summary>
– return information for multiple activities
- POST <https://api.gdms.assetia.cloud/floodrisks/workflow/stages>
– return information for multiple workflow stages
- GET <https://api.gdms.assetia.cloud/floodrisks/picklists/{picklistGroup}>
– return the permitted values and IDs for a picklist field, where {picklistGroup} identifies the field, e.g. “fundingRoutes”. “workflowStages”

12.3 Worked examples

12.3.1 Return flood risk details for one sub-catchment with a known GUID

Example: return flood risk data for a sub-catchment with GUID “123456c4-b3d3-295d-794e-bfd14ebb3dae”.

Endpoint: GET <https://api.gdms.assetia.cloud/floodrisks/floodRisk/summary/123456c4-b3d3-295d-794e-bfd14ebb3dae>

This endpoint returns (shortened):

```
{
  "subCatchment": "SC01234_01",
  "areaId": "00844bc1-870c-4c56-9f7c-f949b3eaf81e",
  other flood risk properties...,
  "subCatchmentLength": 1234.51,
  "floodRiskLastCalculated": "2023-12-28T00:00:01.107",
  "floodRiskStatusId": "96d195bf-3887-4f13-b20f-ec75386c7940",
```

```

    "floodRiskStatus": "X (Risk Addressed)",
    other flood risk properties...
}

```

12.3.2 Return all workflows for one sub-catchment with a known GUID

Example: return all workflows associated with a sub-catchment with GUID "123456c4-b3d3-295d-794e-bfd14ebb3dae".

Endpoint: POST <https://api.gdms.assetia.cloud/floodrisks/workflow/summary>

Payload:

```

{
  "subcatchmentId": "123456c4-b3d3-295d-794e-bfd14ebb3dae",
}

```

Response:

```

{
  "results": [
    {
      "workflow": 123,
      "subCatchment": "SC01234_01",
      "areaId": "00844bc1-870c-4c56-9f7c-f949b3eaf81e",
      "area": 14,
      "road": "A1",
      "subCatchmentLength": 6543.51,
      "startDate": "2021-06-18T09:31:02.08",
      "finishedDate": "2023-03-31T09:50:29",
      other workflow properties...
    },
    {
      "workflow": 456,
      "subCatchment": "SC01234_01",
      "areaId": "00844bc1-870c-4c56-9f7c-f949b3eaf81e",
      "area": 14,
      "road": "A1",
      "subCatchmentLength": 6543.51,
      "startDate": "2019-09-06T10:17:21.173",
      "finishedDate": "2023-03-31T10:04:50",
      other workflow properties...
    }
  ],
  "count": 2,
  "total": 2
}

```

12.3.3 Return workflows by start date and status for one sub-catchment with a known GUID

Example: return the 3 most recent (by start date) workflows that are "In Progress" for a sub-catchment with GUID "94e1e140-1ed6-c37c-b165-654321a20618".

Endpoint: POST <https://api.gdms.assetia.cloud/floodrisks/workflow/summary>

Payload:

```
{
  "limit": 3,
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "OR",
        "filters": [
          {
            "field": "workflowStatus",
            "not": false,
            "operator": "Equals",
            "value": "In Progress"
          }
        ]
      }
    ]
  },
  "sort": [
    {
      "field": "startDate",
      "direction": "desc"
    }
  ],
  "subCatchmentId": "94e1e140-1ed6-c37c-b165-654321a20618"
}
```

Note: "limit" is set to 3 and the sort order is set with the "startDate" field in descending order (most recent first).

This endpoint will return (shortened):

```
{
  "results": [
    {
      "workflow": 17,
      "startDate": "2024-09-24T19:19:18.443",
      "workflowStatus": "In Progress",
      "workflowId": "1e515c9a-eb46-434e-a1c9-9a456a91f0ab",
      "subCatchmentId": "94e1e140-1ed6-c37c-b165-654321a20618",
      other workflow details...
    },
    {
      "workflow": 16,
      "startDate": "2024-01-03T12:12:06.503",
      "workflowStatus": "In Progress",
      "workflowId": "070a4ca9-e2f8-406f-8fec-bef770d94def",
      "subCatchmentId": "94e1e140-1ed6-c37c-b165-654321a20618",
      other workflow details...
    },
    {

```

```

        "workflow": 15,
        "startDate": "2023-03-16T17:00:17.523",
        "workflowStatus": "In Progress",
        "workflowId": "44d8cd94-3f36-4e06-a722-0205958f8df6",
        "subCatchmentId": "94e1e140-1ed6-c37c-b165-654321a20618",
        other workflow details...
    }
],
"count": 5,
"total": 5
}

```

12.3.4 Return details of one workflow with a known GUID

Example: return data for the workflow with GUID "37a47b31-d563-b222-474b-546d995cc41c".

Endpoint: GET <https://api.gdms.assetia.cloud/floodrisks/workflow/summary/37a47b31-d563-b222-474b-546d995cc41c>

This endpoint will return:

```

{
  "workflow": 123,
  "subCatchment": "SC01234_01",
  other workflow properties...,
  "workflowId": "37a47b31-d563-b222-474b-546d995cc41c",
  "subCatchmentId": "123456c4-b3d3-295d-794e-bfd14ebb3dae",
  "relatedActivityId": "e3a79f45-29ac-1cd3-3de4-014790e0976d",
  other workflow properties...
}

```

12.3.5 Return all related activities for one workflow with a known GUID

Example: return all activity data for workflow ID "37a47b31-d563-b222-474b-546d995cc41c".

Endpoint: POST <https://api.gdms.assetia.cloud/floodrisks/workflow/activity/summary>

Payload:

```

{
  "workflowId": "37a47b31-d563-b222-474b-546d995cc41c"
}

```

This will return (shortened):

```

{
  "results": [
    {
      "visibleId": 1234,
      "date": "2023-03-31T09:50:29",
      "workflow": 123,
      "workflowStage": "Other - Desk study complete - field study required (from migrated hotspot)",
      other activity properties...,
      "subCatchment": "SC01234_01",
      other activity properties...,
      "activityId": "e3a79f45-29ac-1cd3-3de4-014790e0976d",
      "workflowId": "37a47b31-d563-b222-474b-546d995cc41c",

```

```
        "subCatchmentId": "123456c4-b3d3-295d-794e-bfd14ebb3dae"
      }
    ],
    "count": 1,
    "total": 1
  }
```

The most useful information returned by this endpoint will be the “date” and “workflowStage”. All workflows should include at least one activity, with the most recent activity’s stage corresponding with the workflow’s current stage.

13 Spills API

Full Swagger documentation for the Spills API is included in the accompanying Spills.json file.

In the examples in this section, where an output would show a total number of records this is shown as “###”.

13.1 Principal entities

The Spills module contains the following entity:

- Spill

13.2 Principal endpoint structure

These are the main types of endpoint for this API:

- GET <https://api.gdms.assetia.cloud/spills/spill/{id}>
 - return information for one spill, where {id} is the record’s GUID
- POST <https://api.gdms.assetia.cloud/spills/spill/{action}>
 - return information for one or multiple spills where {action} is “query”, “summary” or “history”
- GET <https://api.gdms.assetia.cloud/spills/picklists/{picklistGroup}>
 - return the permitted values and IDs for a picklist field, where {picklistGroup} identifies the field, e.g. “cleanupcosts”, “materialtypes”
- POST <https://api.gdms.assetia.cloud/spills/csv/spills>
 - return the spills data in CSV format

13.3 Worked examples

13.3.1 Return spills in a geographical area

Example: return spills located in a rectangular area around the M62 Junction 29. The bounding box coordinates are 431,500E 427,000N; 433,000E 425,000N.

Endpoint: POST <https://api.gdms.assetia.cloud/spills/spill/summary/query>

This example will define the area of interest as a closed polygon in WKT notation, for an example of how to search by geographical area using only filter criteria, see Section 4.3.3.2.

Payload:

```
{
  "searchGeometry": "POLYGON ((431500 427000, 433000 427000, 433000 425000, 431500 425000, 431500 427000))"
}
```

Note that the polygon must be closed so for a rectangle there are five pairs of coordinates and the last coordinate pair (431500 427000) is the same as the first. All coordinates must be to OS grid.

This endpoint returns the data for spills within the specified polygon (shortened):

```
{
  "results": [
    {
      "id": " 29ba6bfd-4abc-d2c1-27e1-c12364e95f8e",
      "visibleId": 1,
      other spills properties...
    },
  ],
}
```

```

{
  "id": " 5d67cf72-8abc-ca07-4dba-fefb123e4054",
  "visibleId": 2,
  other spills properties...
},
other spill results...
],
"count": #,
"total": #
}

```

13.3.2 Return spills by status and road

Example: Return spills data with a status of “Open” on the M6.

Endpoint: POST <https://api.gdms.assetia.cloud/spills/spill/summary/query>

Payload:

```

{
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "OR",
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "road",
            "value": "M6"
          }
        ]
      },
      {
        "condition": "OR",
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "status",
            "value": "Open"
          }
        ]
      }
    ]
  },
}

```

The above “filter” object is equivalent to the following SQL:

WHERE status = 'Open' AND road = 'M6'

This endpoint returns (shortened):

```
{
  "results": [
    {
      "id": "5ec2225f-ee40-e3ed-a30b-abc8d7878e19",
      "visibleId": 123,
      "easting": 3###14.000000,
      "northing": 3###22.000000,
      other spills properties...
    },
    other spills...
  ],
  "count": #,
  "total": ###
}
```

Alternatively, to searching for spills by road name, spills for a catchment can be returned by using the "catchmentVisibleId" or "catchmentId" (GUID) field in the filter criteria.

14 Projects API

Full Swagger documentation for the Projects API is included in the accompanying Projects.json file.

In the examples in this section, where an output would show a total number of records this is shown as “###”.

14.1 Principal entities

The Projects module contains the following principal entities:

- Project
- Project Link (relationship between one Project and one record (“entity”) that is linked to it)

14.2 Principal endpoint structure

These are the main types of endpoint for this API:

- POST <https://api.gdms.assetia.cloud/projects/{action}>
 - return information for multiple projects where {action} is “summary/query”, “summary/csv” or “history”
- POST <https://api.gdms.assetia.cloud/projects/summary/{entity}>
 - return summary information for multiple entities and projects where {entity} is “drawings”, “reports”, “drainage”, “inventoryitem”, “conditionsets”, “geotechnicalasset”, “floodriskworkflow” or “geodatasets”
- POST <https://api.gdms.assetia.cloud/projects/summary/links/query>
 - returns a summary of all existing links for a single project
 - the payload for this request takes the project ID as a parameter to return all entities (e.g. reports, geodatasets etc.) that have been linked to a single project
- GET <https://api.gdms.assetia.cloud/projects/picklists/{picklistGroup}>
 - return the permitted values and IDs for a picklist field, where {picklistGroup} identifies the field, e.g. “projectType”, “linkTypes”

14.3 Worked examples

14.3.1 Return data for a single project with a known GUID

Example: Return data for a project with GUID “83a754ff-a6d2-f32a-be3c-797e50d83ce3”.

Endpoint: POST <https://api.gdms.assetia.cloud/projects/summary/query>

Payload:

```
{
  "projectsIds": ["83a754ff-a6d2-f32a-be3c-797e50d83ce3"]
}
```

Response (shortened):

```
{
  "results": [
    {
      "projectID": 64,
      "projectTitle": "South Coast M27 J4-11",
      "other project properties..."
    }
  ]
}
```

```

    ],
    "count": 1,
    "total": 1
}

```

14.3.2 Return a summary of drawing sets linked to a project with a known GUID

Example: return a drawing sets summary for a project with GUID "83a754ff-a6d2-f32a-be3c-797e50d83ce3".

Endpoint: POST <https://api.gdms.assetia.cloud/projects/summary/drawings>

Payload:

```

{
  "projectsIds": [
    "83a754ff-a6d2-f32a-be3c-797e50d83ce3"
  ]
}

```

Returns (shortened):

```

{
  "results": [
    {
      "visibleID": 1941,
      "projectID": "83a754ff-a6d2-f32a-be3c-797e50d83ce3",
      "drawingSetID": "e47781f8-a39e-18b3-0aba-21c9b3b8d854",
      "drawingSetRef": "03_1004",
      "title": "Title of Drawing Set",
      other drawing set summary properties...
    },
    {
      "visibleID": 1945,
      "projectID": "83a754ff-a6d2-f32a-be3c-797e50d83ce3",
      "drawingSetID": "744f36b6-c186-af0b-89b6-960dcff64821",
      "drawingSetRef": "03_1005",
      other drawing set summary properties...
    },
    {
      "visibleID": 1946,
      "projectID": "83a754ff-a6d2-f32a-be3c-797e50d83ce3",
      "drawingSetID": "43875fa4-210a-fd5d-894a-fb21cd0cf2a8",
      "drawingSetRef": "03_1006",
      other drawing set summary properties...
    },
    other drawing sets...
  ],
  "count": ##,
  "total": ##
}

```

To get full drawing set details, use the "summary/query" endpoint from the Drawings module (15.2) with the drawing set ID(s) of interest provided in the payload as an array of strings.

Endpoint: POST <https://api.gdms.assetia.cloud/drawings/summary/query>

Payload:

```
{
  "drawingSetIds": [
    drawing set ID(s) here...
  ]
}
```

14.3.3 Determine all entities linked to a single project with a known GUID

Example: find the linked entities for a project with GUID "8bea8082-1aa2-e713-761b-b45fbd2180d1".

Endpoint: POST <https://api.gdms.assetia.cloud/projects/summary/links/query>

Payload:

```
{
  "projectId": "8bea8082-1aa2-e713-761b-b45fbd2180d1"
}
```

Returns (shortened):

```
[
  {
    "projectId": "8bea8082-1aa2-e713-761b-b45fbd2180d1",
    "projectNo": 57,
    "entityId": "60af2db7-9cc0-ea58-708a-2d66779748c3",
    "entityTypeId": "96f4499c-08fb-485d-9749-08f19786dc24",
    "entityType": "DrawingSet",
    other entity properties...
  },
  other entities...
]
```

The "entityType" can be used to determine which other endpoints to call for further information about each linked entity, along with the "entityId".

14.3.4 Determine the total number of entities linked to a single project with a known GUID

Example: find the number of linked entities for a project with GUID "8bea8082-1aa2-e713-761b-b45fbd2180d1".

To return the number of entities for a given project, the "entity/count/{id}" endpoint is used.

Endpoint: GET <https://api.gdms.assetia.cloud/projects/entity/count/8bea8082-1aa2-e713-761b-b45fbd2180d1>

A GET request to this endpoint returns 7.

15 Drawings API

Full Swagger documentation for the Drawings API is included in the accompanying DrawingSets.json file.

In the examples in this section, where an output would show a total number of records this is shown as “###”.

15.1 Principal entities

The Drawings module contains the following principal entities:

- Drawing Set
- Located File

15.2 Principal endpoint structure

These are the main types of endpoint for this API:

- POST <https://api.gdms.assetia.cloud/drawings/summary/{action}>
 - return information for multiple drawing sets where {action} is “query” or “csv”
- POST <https://api.gdms.assetia.cloud/drawings/locatedfiles/summary/query>
 - return information for multiple located files
- GET <https://api.gdms.assetia.cloud/drawings/export/locatedfiles/{id}>
 - export located files for a single drawing set where {id} is the ID of the drawing set

15.3 Worked examples

15.3.1 Return data for all located files within a geographical area

Example: return information on all located files in a rectangular area around the M25 Junction 7. The bounding box coordinates are 529,600E 153,900N; 532,000E 152,500N.

Endpoint: POST <https://api.gdms.assetia.cloud/drawings/locatedfiles/summary/query>

Payload:

```
{
  "searchGeometry": "POLYGON ((529600 153900, 532000 153900, 532000 152500, 529600 152500, 529600 153900))"
}
```

Response (shortened):

```
{
  "results": [
    {
      "locatedFileID": "b83e6859-8286-cb14-3351-decac36a0557",
      "visibleId": 8737,
      "drawingSetId": "e178e82e-bdba-03fe-fb2b-c2df929836df",
      other Located file properties...
    },
    {
      "locatedFileID": "d62f1bb6-6600-4340-7707-4cdf4967677c",
      "visibleId": 8738,
      "drawingSetId": "e178e82e-bdba-03fe-fb2b-c2df929836df",
      other Located file properties...
    }
  ]
}
```

```

    other located files...
  ],
  "count": ##,
  "total": ##
}

```

15.3.2 Return drawing sets by road and record type

Example: return all as-built drawing sets for the A4.

Endpoint: POST <https://api.gdms.assetia.cloud/drawings/summary/query>

Payload:

```

{
  "filter": {
    "condition": "AND",
    "filterGroups": [
      {
        "condition": "OR",
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "road",
            "value": "A4"
          }
        ]
      },
      {
        "condition": "OR",
        "filters": [
          {
            "operator": "Equals",
            "not": false,
            "field": "recordType",
            "value": "As-built"
          }
        ]
      }
    ]
  },
}

```

Response (shortened):

```

{
  "results": [
    {
      "drawingSetId": "68e50039-1d2b-da41-2a3a-42162633767e",
      "visibleId": 634,
      "reference": "02_2008",
      "title": "LONDON BRISTOL TRUNK ROAD",
      other drawing set properties...
    }
  ]
}

```

```
    },  
    other drawing sets...  
  ],  
  "count": ##,  
  "total": ##  
}
```