

# Beacon v2 Documentation

---

None

*Beacon API Development Team*

© Copyright 2021-2022, Beacon v2 Documentation Contributors

## Table of contents

---

1. Welcome to the Beacon v2 Documentation	3
1.1 Scope and Purpose	3
2. Informations for Different Types of Beacon Users	4
2.1 Users	4
2.2 Deployers and Implementers	4
2.3 Implementer	5
2.4 Stakeholder	5
3. TODO, Bugs & Changes	6
3.1 TODO	6
3.2 Changes	6
4. Beacon Types and Examples	9
4.1 Beacon "Flavours"	9
4.2 Genomic Variant Queries	10
4.3 Implementations and Examples	13
5. Implement & Deploy a Beacon	14
5.1 Implement and Deploy a Beacon	14
5.2 Formats, Standards and Integrations	15
6. Beacon Components	16
6.1 Beacon v2 Framework	16
6.2 beacon-v2-Models	20
6.3 Filters	21
6.4 Beacon REST API	22
6.5 Terms List	23
7. Schemas	27
7.1 Analyses	27
7.2 Biosamples	28
7.3 Cohorts	31
7.4 Datasets	34
7.5 Genomic Variations	35
7.6 Individuals	39
7.7 Runs	42

# 1. Welcome to the Beacon v2 Documentation

---

## Under Development

Note that Beacon v2 is still [under development](#) and needs to be approved by the [GA4GH](#).

The core documentation (i.e. this document) can be found on [here](#).

## 1.1 Scope and Purpose

---

With growing interest from the community in the implementation of the Beacon protocol into resources and workflows, the major 2.0 release scheduled for Spring 2022 will introduce new features which were considered important by the community such as:

- extended and clearer specified genomic variation queries, including wildcard and region queries (i.e. returning variants within a genomic/chromosomal region)
- get a list of samples related to a phenotype, provided the required authentication or authorization
- powerful *filters*, primarily based on CURIE terms for ontologies and references, including options to control the use of hierarchical terms or the precision of term matching
- scoped data delivery (e.g. matched variant details or sample information) as part of Beacon responses or through *handover* protocols

## 2. Informations for Different Types of Beacon Users

---

The Beacon documentation provides information for different types of stakeholders, depending on their interests and use cases. Although those will overlap, we highlight information relevant for some general scenarios throughout the documentation.

### 2.1 Users

---

A Beacon **user** is interested in querying Beacon instances and networks, either through web interfaces by using the Beacon API. While users of Beacon web forms in principle do not need to understand the underlying query syntax and response formats they too may benefit from some insights into the general capabilities of the underlying protocol.



- Beacon v2 [Model](#)
- Knowing what is available in an instance
- Entry Types
- Datasets & Cohorts
- Granularity & security
- Protocol/Query basics
- Requests, responses & errors
- OpenAPI
- Using Beacon v2 Features
- [Filters](#)
- Alternative schemas

### 2.2 Deployers and Implementers

---

A Beacon **Deployer** is someone who wants to make their genomics resource accessible through the Beacon protocol, without necessarily being interested or experienced in the computational aspects; while a Beacon **Implementer** provides the technical expertise (and potentially may get involved with Beacon development itself, e.g. to extend the protocol for novel use cases).



- Beacon v2 [Models](#)
- Reference Implementation
- Infrastructure requirements
- How to install
- Configuration
- Cohorts and/or Datasets
- Entry types
- Filtering terms
- Alternative schemas
- Granularity & Security
- Administration
- Testing the instance

## 2.3 Implementer

---

### Implementer

- Beacon v2 [Models](#)
- Protocol basics
- Requests, responses & errors
- OpenAPI
- Beacon v2 Features
- Filters
- Alternative schemas
- Configuration
- Granularity & security
- Verifying compliance

## 2.4 Stakeholder

---

### Stakeholder

- Integration into GA4GH
- Leveraging The Beacon Framework in other domains
- Success Stories:
- [Implementations](#)
- Real world data

## 3. TODO, Bugs & Changes

---

### 3.1 TODO

---

#### 3.1.1 Documentation

---

- content for [Standards Integration](#) (as noted on page)
- re-structuring of [Framework page](#)
- re-structuring of [Models page](#)
- integration with the Schemas pages
- removal of the individual schemas from the main navigation
- delete documentation in framework and model repo READMEs and point here
- fix https (probably @mbaudis has to do some registrar configuration...)
- content for the [Filters page](#)
- merging (?) of the [Introduction](#) and [Documentation Scopes](#) pages
- clarification, re-structuring, links ...
- add more [Implementations](#)
- extend [Query documentation](#), also using content from the variant scouts document

#### 3.1.2 Repositories

---

- name and title change
- adjusting all the links accordingly
- UG: fix transfer of examples
- retiring of framework and model repos

### 3.2 Changes

---

#### 2002-03-18: Macros and Variables for Documentation pages

The `mkdocs-macros-plugin` has been activated, allowing the use of site-wide variables:

- `repo_model_url`: `https://github.com/ga4gh-beacon/beacon-v2-unity-testing/tree/main/models/src`
- this can be used inline as `https://github.com/ga4gh-beacon/beacon-v2-unity-testing/tree/main/models/src`

#### 2002-03-16: Documentation Content and Formats Updates

- addition of more variant query examples
- new landing pages for [Implementations and Networks](#) and [Standards Integration](#)
- many adjustments to documentation structure, appearance and representation (e.g. content tabs for query examples)

### 2022-03-14: Documentation in Repository

As of today the new/emerging Beacon v2 documentation is maintained in this repository. We're testing rendered versions (same text/code base) through Github actions ([here](#)) and [ReadTheDocs](#).

- ✔️sting of [ReadTheDocs version](#) vs. a [material](#) themed build
- ✔️reated and linked [docs.genomebeacons.org](#) sub-domain to the Github hosted version of the rendered documentation
- ✔️erging of previous separate documentation repository content from *beacon-v2-schema-documentation* in the "unity" repository and archiving of the old one

### 2022-03-11: Removing `yaml` export version

Since moving to having the source in YAML the existence of a separate `yaml` export seems unnecessary & maybe confusing. Removed.

### 2022-03-09: Nesting models

The structure of the `models` directory has now be changed to have the default model as one of possibly multiple options as per the discussions in [#1](#). The current structure (below) might not be final (e.g. placing of the `beaconConfiguration.yaml`, `beaconMap.yaml`, `endpoints.yaml` files?).

```
beacon
|
|-- framework ...
|-- models
|   |-- src
|   |   |-- beacon-v2-default-model
|   |   |   |-- analyses ...
|   |   |   |-- biosamples ...
|   |   |   |-- genomicVariations ...
|   |   |   |-- ...
|   |   |   |-- endpoints.yaml
|   |-- json
|   |   |-- beacon-v2-default-model
|   |   |   |-- analyses ...
|   |   |   |-- biosamples ...
|   |   |   |-- genomicVariations ...
|   |   |   |-- ...
|   |   |   |-- endpoints.yaml
|-- bin ...
|-- docs ...
...
```

### 2022-03-08: Automated pulling from current origin repos

- added simple pull commands to the conversion for automatic update to the donor repos

```
git -C $BEACONMODELPATH pull
git -C $BEACONFRAMEWORKPATH pull
```

- updated to current crop of PRs

### 2022-02-24: Path fixes

- changed the path replacements to the current repo, resulting in e.g. [raw.githubusercontent.com/ga4gh-beacon/beacon-v2-unity-testing/main/framework/json/responses/sections/beaconInformationalResponseMeta.json](#)

### 2022-02-23: Re-tool

- ✔️placement of the previopus general `yamler.py` with a dedicated `beaconYamler.py`
- ✔️oving replacements to [bin/config.yaml](#)
- ✔️quirement for complete arguments (in and out paths, in- and out formats) - see [bin/yamlerRunner.sh](#)

**2022-02-22: Creation of repository**

- ✔ design of directory structure
- ✔ test tooling & population with auto-converted files



## 4. Beacon Types and Examples

---

### 4.1 Beacon "Flavours"

---

While the original Beacon v1 only provided Boolean (*i.e.* **YES/NO**) responses on queries for the existence of specific genomic variants, Beacon v2 is a flexible protocol that supports different usage scenarios - also called "flavours", since they are more a representation of usage types w/o prescribing their specific details.

Importantly, the Beacon framework separates query options from the response side. In that way a privacy-protecting<sup>1</sup> Boolean Beacon still may offer more query features - and therefore better usability - compared to the first Beacon concept implementations 😊

For detailed information about the technical implementation of the different logical scopes please see the [Framework](#) documentation.

#### 4.1.1 Boolean Response Beacon

---

A *Boolean* Beacon is in its response similar to Beacon v1 - *i.e.* responding with a *true* or *false* value when queried for the existence of some data in a resource.

However, in contrast to earlier versions, in Beacon v1 *in principle* a "Boolean Beacon" may implement all types of query options (e.g. combinations of various filters and genomic query parameters) but still offer a Boolean response, either as sole option or depending on the user's authentication status.

#### Boolean Response

#### Beacon Count Response

#### 4.1.2 Beacons Supporting Data and Information Delivery

---

#### Beacon Data Model

#### Alternative Data Models

#### Data Handover

---

---

<sup>1</sup>. Privacy protecting as in "reasonably protecting by design but not immune to complex re-identification attacks". ↩

## 4.2 Genomic Variant Queries

For querying of genomic variations Beacon v2 builds on and extends the options provided by earlier versions.

### 4.2.1 Query Examples

#### Beacon SNV Query

EXAMPLE: *EIF4A1* SINGLE BASE MUTATION

This is an example for a single base mutation ( G>A ) in the *EIF4A1* eukaryotic translation initiation factor 4A1.

Beacon v2 GET	Beacon v2 POST	Beacon v1	Beacon v0.3
---------------	----------------	-----------	-------------

?referenceName=NC\_000017.11&start=7577120&referenceBases=G&alternateBases=A

#### OPTIONAL

- datasetIds=\_\_some-dataset-ids\_\_
- filters ...

```
{
  "$schema": "beaconRequestBody.json",
  "meta": {
    "apiVersion": "2.0",
    "requestedSchemas": [
      {
        "entityType": "genomicVariation",
        "schema": "https://raw.githubusercontent.com/ga4gh-beacon/beacon-v2-unity-testing/main/models/json/beacon-v2-default-model/genomicVariations/defaultSchema.json"
      }
    ]
  },
  "query": {
    "requestParameters": {
      "referenceName": "NC_000017.11",
      "start": [7577120],
      "referenceBases": "G",
      "alternateBases": "A"
    }
  },
  "requestedGranularity": "record",
  "pagination": {
    "skip": 0,
    "limit": 5
  }
}
```

There are optional parameters [ datasetIds , filters ...] and also the option to specify the response type (through requestedGranularity ) and returned data format ( requestedSchemas ). Please follow this up in the [framework documentation](#).

?assemblyId=GRCh38&referenceName=17&start=7577120&referenceBases=G&alternateBases=A

#### OPTIONAL

- datasetIds=\_\_some-dataset-ids\_\_

?ref=GRCh38&chrom=17&pos=7577121&referenceAllele=C&allele=A

#### OPTIONAL

- beacon=\_\_some-beacon-id\_\_

Before Beacon v0.4 a 1-based coordinate system was being used.

## Beacon CNV Queries

EXAMPLE: *TP53* DELETION QUERY BY COORDINATES

The following example shows a "bracket query" for focal deletions of the *TP53* gene locus:

- The start of the deletion has to occur anywhere from approx. 2.5Mb 5' of the CDR start to just before the end of the CDR.
- The end of the matched CNVs has to be anywhere from the start of the gene locus to approx. 2.5Mb 3' of its end.

This leads to matching of deletion CNVs which have at least some base overlap with the gene locus but are not larger than approx. 5Mb (operational definitions of focality vary between 1 and 5Mb).

Beacon v2 GET	Beacon v2 POST	Beacon v1	Beacon v0.3
?datasetIds=TEST&referenceName=NC_000017.11&variantType=DEL&start=5000000&start=7676592&end=7669607&end=10000000			

### OPTIONAL

- datasetIds=\_\_some-dataset-ids\_\_
- filters ...

```
{
  "$schema": "https://raw.githubusercontent.com/ga4gh-beacon/beacon-v2-unity-testing/main/framework/json/requests/beaconRequestBody.json",
  "meta": {
    "apiVersion": "2.0",
    "requestedSchemas": [
      {
        "entityType": "genomicVariation",
        "schema": "https://raw.githubusercontent.com/ga4gh-beacon/beacon-v2-unity-testing/main/models/json/beacon-v2-default-model/genomicVariations/defaultSchema.json"
      }
    ]
  },
  "query": {
    "requestParameters": {
      "referenceName": "NC_000017.11",
      "start": [ 5000000, 7676592 ],
      "end": [ 7669607, 10000000 ],
      "variantType": "DEL"
    }
  },
  "requestedGranularity": "record",
  "pagination": {
    "skip": 0,
    "limit": 5
  }
}
```

There are optional parameters [ datasetIds , filters ...] and also the option to specify the response type (through requestedGranularity ) and returned data format ( requestedSchemas ). Please follow this up in the [framework documentation](#).

?assemblyId=GRCh38&referenceName=17&variantType=DEL&start=5000000&start=7676592&end=7669607&end=10000000
--

### OPTIONAL

- datasetIds=\_\_some-dataset-ids\_\_

CNV query options were only implemented with Beacon v0.4, based on Beacon<sup>+</sup> prototyping.

(TBD)

## 4.2.2 Query Parameter Change Log

### Beacon v2

- use of sequence reference id's which obviate the need for a assemblyId parameter

- **range queries**

- with specified single start and end parameters a query should match any variant with partial or complete overlap with this sequence range
- additional parameters (e.g. `referenceBases`, `alternateBases`, `variantType`...) may be used to scope the range query

**Beacon v1 (based on v0.4)**

- switch to *0-based interbase* coordinates for the API with *1-based* coordinates recommended for query forms
- this represents the common GA4GH usage and the practice e.g. of the UCSC genome browser
- introduction of *bracketed queries*
- specification of intervals for `start` and `end` positions when querying multi-base variants allows for "fuzzy" CNV queries
- support of a `variantType` parameter to specify e.g. CNV queries ( `DUP`, `DEL` )
- `variantType` is not required for precise queries with specified `referenceBases` and `alternateBases`

## 4.3 Implementations and Examples

---

### 4.3.1 Registry Server

The Beacon registry server, hosted through the European Genome-Phenome Archive, monitors a number of implementations of the Beacon v2 protocol by various organisations actively involved in Beacon protocol development.

Link: [Beacon v2 GA4GH Approval Registry](#)

### 4.3.2 Beacon Networks

(TBD)

### 4.3.3 Example Implementations

---

#### Progenetix API

The Progenetix database and cancer genomic information resource contains genome profiles of more than 140'000 individual cancer genome screening experiments, with the majority representing results from genomic copy number assessment studies. With its Beacon<sup>+</sup> forward-looking test implementation, since 2016 Progenetix has been developing concepts for Beacon protocol extensions such as CNV query options or handover data delivery.

Link: [Documentation page](#) for Progenetix REST paths

## 5. Implement & Deploy a Beacon

---

### 5.1 Implement and Deploy a Beacon

---

Owners and managers of genomic data resources may (should!) be interested in "beaconizing" their resources through the implementation of a Beacon API[1] for web-protocol based access to their data.

Beacon software can be developed *de novo*, make use of code from existing solutions or can directly be derived from the [Beacon reference implementation](#) provided by the beacon development team.

#### 5.1.1 Topics of Interest

---

1. Beacon "flavours"
2. Error handling
3. Features to implement
4. Verifying compliance
5. Versioning
6. Granularity & security
  - Protocol details and flow

#### 5.1.2 Potential Documents

---

1. [Beacon v2 Models](#)
2. Protocol basics
  - Requests, responses & errors
  - OpenAPI
3. Beacon v2 Features
  - [Genomic Queries](#)
  - [Filters](#)
  - Alternative schemas
4. Configuration
  - Granularity & security
5. Verifying compliance

- 
1. Application Programming Interface ←

## 5.2 Formats, Standards and Integrations

---

### 5.2.1 Data Formats and Standards

---

#### Schema Language and Conventions

JSON Schema, YAML/JSON, camelCasing and others ...

#### Genome Coordinates

0-based interbase etc., see [schemablocks.org](https://www.schemablocks.org)

#### Dates and Times

ISO8601 etc., see [schemablocks.org](https://www.schemablocks.org)

### 5.2.2 Integration with External Standards

---

The development of the Beacon v2 framework and default model closely follows and widely adopts concepts and schemas from approved GA4GH products such as Phenopackets and the Variant Representation Standard (VRS).

(TBD)

## 6. Beacon Components

---

### 6.1 Beacon v2 Framework

---

#### 6.1.1 Introduction

The GA4GH Beacon specification is composed by two parts:

- the Beacon Framework (in *this* repo)
- the Beacon Model (in the [Models repo](#))

The **Beacon Framework** is the part that describes the overall structure of the API requests, responses, parameters, the common components, etc. It could also be referred in this document as simply the *Framework*.

A **Beacon Model** describes the set of concepts included in a Beacon version (e.g. Beacon v2), like *individual* or *biosample*. It could also be referred in this document as simply the *Model*.

The Framework could be considered the *syntax* and the Model as the *semantics*.

Refer to the [Models repo](#) for further information about the Model and how to use it.

The Framework doesn't include anything related to specific entities but only the mechanisms for querying them and parsing the responses. The BF is, therefore, independent from/agnostic to any specific Model. It can be leveraged to describe models from other domains like proteomics, imaging, biobanking, etc.

A **Beacon instance** is just an implementation of a Beacon Model that follows the rules stated by the Beacon Framework.

If you are a Beacon implementer, then, you don't need to clone this (Framework) repo, you only need to **copy** (or *clone*) the Beacon Model and modify it to your specific instance. You will find plenty of references to the Framework in the Model copy, and you will use the Json schemas in the Framework to validate that both the structure of your requests and responses are compliant with the Beacon Framework. The [Beacon verifier](#) tool would help in such validation.

The Framework repo includes the elements that are common to all Beacons:

1. The configuration files
2. The Json schemas for the requests, the responses, and its respective sections
3. The files of every Beacon root
4. Examples of all the above (using a fake and simple Model)

#### Coding and naming conventions

For historical reasons, in the names of entities, parameters and URLs we are following the conventions: \* Entity names:

`PascalCase` \* parameters: `camelCase` \* URI path elements: `snake_case` The only exception is: `service-info` which is a required GA4GH standard and has a different word separation convention.

#### 6.1.2 Folder structure in the framework repo

---

The above listed elements are organized in several folders (*in alphabetical order*):

- **common:** Json schemas and examples of the components used in other parts of the specification.
- **configuration:** Json schemas and examples for the configuration files that every Beacon MUST implement.
- **requests:** Json schemas and examples for the different sections of a request.
- **responses:** Json schemas and examples for the different types of responses and response sections.
- **root folder:** It only includes the definition of the Beacon root endpoints.



## The root folder and the Beacon root endpoints

The *root* folder only contains the `endpoints.json` document, an OpenAPI 3.0.2 description of the endpoints that every Beacon instance MUST implement. The endpoints are: \* the *root* (`/`) and */info* that MUST return information (metadata) about the Beacon service and the organization supporting it. \* the */service-info* endpoint that returns the Beacon metadata in the GA4GH Service Info schema. \* the */configuration* endpoint that returns some configuration aspects and the definition of the entry types (e.g. *genomic variants*, *biosamples*, *cohorts*) implemented in that specific Beacon server or instance. \* the */entry\_types* endpoints that only return the section of the configuration that describes the entry types in that Beacon. \* the */map* endpoint that returns a map (like a web *sitemap*) of the different endpoints implemented in that Beacon instance. \* the */filtering\_terms* endpoint that returns a list of the filtering terms accepted by that Beacon instance.

Most of these endpoints simply return the configuration files that are in the Beacon configuration folder. Of course, every Beacon instance would have their particular instance of such documents, including the configuration of such instance.

*Note:* It could be argued that the Beacon configuration files are different for every Beacon instance and, hence, they should be part of the Model. However, the configuration files MUST be used, exactly with the same schema, by *any* model, independently if that Beacon follows the Beacon v2 Model or any other. Additionally, these endpoints and configuration files are *critical* for a Beacon client to be able to understand and use a Beacon instance. Therefore, we have considered it to be an essential part of the Framework and belonging to it.

## The Configuration

Contains the Json schema files that describe the Beacon configuration, its contents are described in the section above, as they have almost a 1-to-1 relationship with such endpoints. Further details about the specific content of each file could be find in the corresponding sections below.

## The Requests

Contains the following Json schemas:

- **beaconRequestBody.json:** Schema for the whole Beacon request. It is named `RequestBody` to keep the same nomenclature used by OpenAPI v3, but it actually contains the definition of the whole HTTP POST request payload.
- **beaconRequestMeta.json:** Meta section of the Beacon request. It includes request context details relevant for the Beacon server when processing the request, like the Beacon API version used to format the request or the schemas expected for the entry types in the response.
- **filteringTerms.json:** defines the schema for the filters included in the request.
- **requestParameters.json** defines the, very free, schema of the parameters included in the request.
- **examples-fullDocuments folder:** includes examples of "actual" requests. The example labelled with `MIN` in the name shows the minimal required attributes for the request to be compliant. The example labelled with `MAX` in the name includes a richer case with all the sections filled in.
- **examples-sections folder:** includes examples of "actual" sections of the requests. It is included to allow specification designers and Beacon implementers to check the compliance with a single section instead of having to implement a whole request. Such way, We aim to facilitate an "incremental" implementation of an instance.

### DIFFERENCES BETWEEN FILTERINGTERMS AND REQUESTPARAMETERS

Both, the filters (*filteringTerms*) and the parameters (*requestParameters*), are used to refine the query. The availability of two mechanisms to refine the queries could sound initially confusing, but that separation is taylorred to facilitate the interpretation of the request by the Beacon server.

An basic difference is that, in HTTP GET requests, each parameters is named (e.g. 'id', 'skip','limit') while filters go under the same named parameter 'filters'. For HTTP POST requests, the difference relays on paramaters having each one a separate definition (e.g. `id` is a `string`, while `skip` is an `integer`), while all filters follow the schema described in `/requests/filteringTerms.json`.

An unrestricted query like `/datasets` should return the list of all datasets in a Beacon instance. That query could be refined by adding a generic condition like: "return only datasets which could be used for 'general research'" or "return only the first 10

datasets". The former belong to the filter category, the latter to the parameters. If you are a beacon implementer, a rule of thumb could be:

- anything that requires its own schema would be a request parameter
- anything that could be represented by an ontology term would go into the filters section.
- anything else would probably be a request parameter.

## The Responses

The Beacon concept includes several types of responses: some informative or informational and some with actual data payloads, and the error one.

### THE INFORMATIONAL RESPONSES

A Beacon is able to return information, details, about itself. Many of the schema responses included in the `responses` folder have a 1-to-1 relationship with the corresponding configuration documents and their equivalent root endpoints, e.g. the `beaconEntryTypeResponse.json` is the schema of a response that wraps the `beaconConfiguration.json` document, and is then used as the payload of the `/entry_types` root endpoint. Schematically: *\* configuration/an\_schema.json*: describes the schema of the configuration file itself. *\* responses/an\_schema\_response.json*: describes the format of the response that returns these configuration information. *\* root/endpoints.json*: describes the API endpoints to be called and parameters to be used to retrieve such responses.

The following schemas refer to informational responses: *beaconConfigurationResponse*, *beaconEntryTypeResponse*, *beaconFilteringTermsResponse*, and *beaconMapResponse*.

### THE RESULTS RESPONSES

A Beacon could return responses at different granularity levels:

- **boolean response:** only returns `exists: true` ('Yes') or `exists: false` ('No') to a given query.
- **count response:** returns `Yes/No` and the number of matching results.
- **resultset response:** returns `Yes/No`, the number of matching results and details of them per every collection (e.g. every dataset or cohort) and, if granted, details on every record that matches the query.

Each of these granularity levels has an equivalent response schema:

- **boolean:** `beaconBooleanResponse`
- **count:** `beaconCountResponse`
- **resultset** (with or w/o record details): `beaconResultSetsResponse`

An additional schema, *beaconCollectionsResponse*, describes such responses that returns details about the collections in a Beacon, but not the collection content themselves. Otherwise said, the response describes a dataset, but not returns the contents of any dataset.

## The common components

Some elements are transversal to the Framework and to any model, e.g. the schema for describing an ontology term or the reference to an external schema (like the reference to GA4GH Phenopackets or GA4GH Service Info schemas).

### Testing the compliance of an implementation with *testMode*

Given that the flexibility allowed in the implementation of each Beacon instance, and the security restrictions that could apply (e.g. only answering after authentication of the user), a mechanism is required for allowing testing the compliance of a Beacon. A first step in this compliance testing is done by the implementer by checking that received requests are correct and that the generated responses match the provided schemas. However, an external compliance testing is desirable when the Beacon instance plans to be integrated in a network or to engage in dialogs with a diversity of clients. For this second scenario, the *testMode* parameter was included.

A Beacon instance could receive a request with the *testMode* parameter activated (value= *true*) in which case the Beacon MUST respond, with actual or fake contents, using the response format and skipping any user authentication. The fact that a response has been generated for testing purposes is included in the meta section of the response.

### 6.1.3 The Beacon Configuration file

The file `/configuration/beaconConfiguration.json` defines the schema (in Json schema draft-07) of the Json file that includes core aspects of a Beacon instance configuration. The schema includes four sections:

1. **\$schema:** that MUST BE a reference to a schema. In the Models, the instances of that file will point to *this file*. Having the schema allows verifying that the document is compliant with it.
2. **maturityAttributes:** Declares the level of maturity of the Beacon instance. Available values are:
  - **DEV:** Service potentially unstable, not using real data, which availability and data should not be used in production setups.
  - **TEST:** The service is expected to be stable, meaning up and available, but does not include real data to be trusted for real world queries.
  - **PROD:** Service stable, at production level standards, containing actual data. Except when testing, most of the Beacon queries are expected to be answered by 'PROD' Beacons.
3. **securityAttributes:** Configuration of the security aspects of the Beacon. By default, a Beacon that does not declare the configuration settings would return `boolean` (true/false) responses, and only if the user is authenticated and explicitly authorized to access the Beacon resources. Although this is the safest set of settings, it is not recommended unless the Beacon shares very sensitive information. Non sensitive Beacons should preferably opt for a `record` and `PUBLIC` combination.
  - **defaultGranularity:** Default granularity of the responses. Some responses could return higher detail, but this would be the granularity by default.
  - **securityLevels:** All access levels supported by the Beacon. Any combination is valid, as every option would apply to different parts of the Beacon. Available options are:

Granularity	Description
<code>boolean</code>	returns 'true/false' responses.
<code>count</code>	adds the total number of positive results found.
<code>aggregated</code>	returns summary, aggregated or distribution like responses per collection.
<code>record</code>	returns details for every row.

For those cases where a Beacon prefers to return records with less, not all, attributes, different strategies have been considered, e.g.: keep non-mandatory attributes empty, or Beacon to provide a minimal record definition, but these strategies still need to be tested in real world cases and hence no design decision has been taken yet.

security level	description
<code>PUBLIC</code>	Any anonymous user can read the data
<code>REGISTERED</code>	Only known users can read the data
<code>CONTROLLED</code>	Only specifically granted users can read the data

#### EXAMPLE

```
"maturityAttributes": {
  "productionStatus": "DEV"
},
"securityAttributes": {
  "defaultGranularity": "boolean",
  "securityLevels": ["PUBLIC", "REGISTERED", "CONTROLLED"]
}
```

The Beacon in the example is in development status, returns boolean answers by default, and has queries available in any of the access levels.


## 6.2 beacon-v2-Models


---

### 6.2.1 Introduction

The GA4GH Beacon specification is composed by two parts:

- the Beacon [Framework](#) 
- the Beacon Models 

The **Beacon Framework** (in Framework repo ) is the part that describes the overall structure of the API requests, responses, parameters, the common components, etc. It could also be referred in this document as simply the *Framework*.

**Beacon Models** (in the Models repo ) describes the set of concepts included in a Beacon version (e.g. Beacon v2), like *individual* or *biosample*, and also the relationships between them. It could also be referred in this document as simply the *Model*.

The Framework could be considered the *syntax* and the Model as the *semantics*.

Refer to the [Framework repo](#) for further information about the Framework and its parts.

A **Beacon instance** is just an implementation of a Beacon Model that follows the rules stated by the Beacon Framework.

If you are a Beacon implementer, then, you don't need to clone the Framework repo, you only need to **copy** (or *clone*) the Beacon Model and modify it to your specific case. You will find plenty of references to the Framework in the Model copy, and you will use the Json schemas there to validate that both the structure of your requests and responses are compliant with the Beacon Framework. The Framework is not used to check the schema in the responses payload (e.g. the actual details of a biosample of a cohort). The schemas for that are included in the Model that you should have copied.

The Model repo points to several hosts the default model for Beacon v2:

1. **The TEMPLATE Model:** [repo](#) is the most basic model. Its purpose is twofold 1) as starting point for any *new* model (so to say, not Beacon v2) and 2) as a learning tool.
2. **The Beacon v2 Model:** (in [Models](#) ) represents the complete Beacon v2 *Default* Model.
3. **The Beacon v1 Model:** [repo](#) Provided as an example for Beacon v1 implementers that want to update to Beacon v2 but not planning to add any additional entry type to their Beacon.

#### Coding and naming conventions

For historical reasons, in the names of entities, parameters and URLs we are following the conventions:

- Entity names: `PascalCase`
- parameters: `camelCase`
- URI path elements: `snake_case` The only exception is: `service-info` which is a required GA4GH standard and has a different word separation convention.

## 6.3 Filters

---

*Filters* represent a powerful addition to the Beacon query API. Briefly, they empower such options as queries for phenotypes, disease codes or technical parameters associated with observed genomic variants.

(TBD)

## 6.4 Beacon REST API

---

While the full power of the Beacon API can be unlocked through the use of structured queries using JSON serialization ("POST" requests), the majority of common queries can be implemented through standard query URLs with parameters (GET queries).

### 6.4.1 Example Beacon API URLs

---

Beacon REST paths in general follow the format

```
__APIroot__/__entryType__/{id}/
```

or

```
__APIroot__/__entryType__/{id}/__requestedSchema__
```

A typical example would e.g. the request to retrieve all genomic variants associated with a biosample

```
https://example.com/beacon/api/biosamples/bios-st4582/g_variants
```

#### ST Endpoint Definitions

The endpoint paths available for a given Beacon instance are defined in `__APIroot__/beaconMap/` [Github](#)

#### Typical Endpoint Patterns

(TBD)

## 6.5 Terms List

---

- [affected](#)
- [age](#)
- [ageAtExposure](#)
- [ageAtOnset](#)
- [ageAtProcedure](#)
- [ageGroup](#)
- [ageOfOnset](#)
- [ageRange](#)
- [aligner](#)
- [alleleFrequency](#)
- [alleleOrigin](#)
- [alternateBases](#)
- [aminoacidChanges](#)
- [analysisDate](#)
- [analysisId](#)
- [annotatedWith](#)
- [assayCode](#)
- [assemblyId](#)
- [availability](#)
- [availabilityCount](#)
- [biosampleId](#)
- [biosampleStatus](#)
- [bodySite](#)
- [caseLevelData](#)
- [category](#)
- [clinicalInterpretations](#)
- [clinicalRelevance](#)
- [clinVarIds](#)
- [cohortDataTypes](#)
- [cohortDesign](#)
- [cohortExclusionCriteria](#)
- [cohortId](#)
- [cohortInclusionCriteria](#)
- [cohortName](#)
- [cohortSize](#)
- [cohortType](#)
- [collectionDate](#)
- [collectionEvents](#)
- [collectionMoment](#)
- [conditionId](#)

- `createDateTime`
- `dataUseConditions`
- `date`
- `dateOfProcedure`
- `description`
- `diagnosticMarkers`
- `disease`
- `diseaseCode`
- `diseaseConditions`
- `diseases`
- `distribution`
- `dose`
- `duoDataUse`
- `duration`
- `effect`
- `end`
- `ethnicities`
- `ethnicity`
- `eventAgeRange`
- `eventCases`
- `eventControls`
- `eventDataTypes`
- `eventDate`
- `eventDiseases`
- `eventEthnicities`
- `eventGenders`
- `eventLocations`
- `eventNum`
- `eventPhenotypes`
- `eventSize`
- `eventTimeline`
- `evidence`
- `evidenceCode`
- `evidenceType`
- `excluded`
- `exposureCode`
- `exposures`
- `externalUrl`
- `familyHistory`
- `featureClass`
- `featureID`
- `featureType`
- `frequencies`



- [frequency](#)
- [frequencyInPopulations](#)
- [genders](#)
- [geneIds](#)
- [genomicFeatures](#)
- [genomicHGVSId](#)
- [geographicOrigin](#)
- [histologicalDiagnosis](#)
- [id](#)
- [identifiers](#)
- [individualId](#)
- [info](#)
- [interventionsOrProcedures](#)
- [iso8601duration](#)
- [label](#)
- [libraryLayout](#)
- [librarySelection](#)
- [librarySource](#)
- [libraryStrategy](#)
- [locations](#)
- [measurements](#)
- [measurementValue](#)
- [measures](#)
- [memberId](#)
- [members](#)
- [modifiers](#)
- [molecularAttributes](#)
- [molecularEffects](#)
- [name](#)
- [notes](#)
- [numSubjects](#)
- [observationMoment](#)
- [obtentionProcedure](#)
- [onset](#)
- [pathologicalStage](#)
- [pathologicalTnmFinding](#)
- [pedigrees](#)
- [phenotypicConditions](#)
- [phenotypicEffects](#)
- [phenotypicFeatures](#)
- [pipelineName](#)
- [pipelineRef](#)
- [platform](#)

- [platformModel](#)
- [population](#)
- [position](#)
- [procedure](#)
- [procedureCode](#)
- [proteinHGVSIds](#)
- [reference](#)
- [referenceBases](#)
- [refseqId](#)
- [resolution](#)
- [role](#)
- [route](#)
- [runDate](#)
- [runId](#)
- [sampleOriginDetail](#)
- [sampleOriginType](#)
- [sampleProcessing](#)
- [sampleStorage](#)
- [severityLevel](#)
- [sex](#)
- [source](#)
- [sourceReference](#)
- [stage](#)
- [start](#)
- [toolName](#)
- [toolReferences](#)
- [transcriptHGVSIds](#)
- [treatmentCode](#)
- [treatments](#)
- [tumorGrade](#)
- [tumorProgression](#)
- [units](#)
- [updateDateTime](#)
- [value](#)
- [variantAlternativeIds](#)
- [variantCaller](#)
- [variantInternalId](#)
- [variantLevelData](#)
- [variantType](#)
- [version](#)
- [zygosity](#)

## 7. Schemas

### 7.1 Analyses

Field	Description	Type	Properties	Example	Enum
<a href="#">aligner</a>	Reference to mapping/alignment software	string	NA	bwa-0.7.8	NA
<a href="#">analysisDate</a>	Date at which analysis was performed.	string	NA	NA	NA
<a href="#">biosampleId</a>	Reference to Biosample ID.	string	NA	S0001	NA
<a href="#">id</a>	Analysis reference ID (external accession or internal ID)	string	NA	NA	NA
<a href="#">individualId</a>	Reference to Individual ID.	string	NA	P0001	NA
<a href="#">info</a>	Placeholder to allow the Beacon to return any additional information that is necessary or could be of interest in relation to the query or the entry returned. It is recommended to encapsulate additional informations in this attribute instead of directly adding attributes at the same level than the others in order to avoid collision in the names of attributes in future versions of the specification.	object	NA	NA	NA
<a href="#">pipelineName</a>	Analysis pipeline and version if a standardized pipeline was used	string	NA	NA	NA
<a href="#">pipelineRef</a>	Link to Analysis pipeline resource	string	NA	NA	NA
<a href="#">runId</a>	Run identifier (external accession or internal ID).	string	NA	SRR10903401	NA
<a href="#">variantCaller</a>	Reference to variant calling software / pipeline	string	NA	GATK4.0	NA

## 7.2 Biosamples

---

Field	description	type	properties
<a href="#">biosampleStatus</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">collectionDate</a>	Date of biosample collection in ISO8601 format.	string	NA
<a href="#">collectionMoment</a>	Individual's or cell culture age at the time of sample collection in the ISO8601 duration format P[n]Y[n]M[n]DT[n]H[n]M[n]S .	string	NA
<a href="#">diagnosticMarkers</a>	Clinically relevant biomarkers. RECOMMENDED.	array	<a href="#">id</a> , <a href="#">label</a>
<a href="#">histologicalDiagnosis</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">id</a>	Biosample identifier (external accession or internal ID).	string	NA
<a href="#">individualId</a>	Reference to the individual from which that sample was obtained.	string	NA
<a href="#">info</a>	Placeholder to allow the Beacon to return any additional information that is necessary or could be of interest in relation to the query or the entry returned. It is recommended to encapsulate additional informations in this attribute instead of directly adding attributes at the same level than the others in order to avoid collision in the names of attributes in future versions of the specification.	object	NA
<a href="#">measurements</a>	List of measurements of the sample.	array	<a href="#">assayCode</a> , <a href="#">date</a> , <a href="#">measurementValue</a> , <a href="#">notes</a> , <a href="#">observation</a>
<a href="#">notes</a>	Any relevant info about the biosample that does not fit into any other field in the schema.	string	NA
<a href="#">obtentionProcedure</a>	Class describing a clinical procedure or intervention.	object	<a href="#">ageAtProcedure</a> , <a href="#">bodySite</a> , <a href="#">dateOfProcedure</a> , <a href="#">procedure</a>
<a href="#">pathologicalStage</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">pathologicalTnmFinding</a>	Pathological TNM findings, if applicable, preferably as subclass of NCIT:C48698 - Cancer TNM Finding Category (NCIT:C48698). RECOMMENDED.	array	<a href="#">id</a> , <a href="#">label</a>
<a href="#">phenotypicFeatures</a>	List of phenotypic abnormalities of the sample. RECOMMENDED.	array	<a href="#">evidence</a> , <a href="#">excluded</a> , <a href="#">featureType</a> , <a href="#">modifiers</a> , <a href="#">notes</a> , <a href="#">onset</a> , <a href="#">severityLevel</a>
<a href="#">sampleOriginDetail</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">sampleOriginType</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>

Field	description	type	properties
<a href="#">sampleProcessing</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">sampleStorage</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">tumorGrade</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">tumorProgression</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>

## 7.3 Cohorts

---

Field	description	type	properties
<a href="#">cohortDataTypes</a>	Type of information. Preferably values from Genomics Cohorts Knowledge Ontology (GeCKO) or others when GeCKO is not applicable.	array	<a href="#">id</a> , <a href="#">label</a>
<a href="#">cohortDesign</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">cohortExclusionCriteria</a>	Criteria used for defining the cohort. It is assumed that all cohort participants will match or NOT match such criteria.	object	<a href="#">ageRange</a> , <a href="#">diseaseConditions</a> , <a href="#">ethnicities</a> , <a href="#">genders</a> , <a href="#">locations</a> , <a href="#">phenotypes</a>
<a href="#">cohortId</a>	Cohort identifier. For 'study-defined' or 'beacon-defined' cohorts this field is set by the implementer. For 'user-defined' this unique identifier could be generated upon the query that defined the cohort, but could be later edited by the user.	string	NA
<a href="#">cohortInclusionCriteria</a>	Criteria used for defining the cohort. It is assumed that all cohort participants will match or NOT match such criteria.	object	<a href="#">ageRange</a> , <a href="#">diseaseConditions</a> , <a href="#">ethnicities</a> , <a href="#">genders</a> , <a href="#">locations</a> , <a href="#">phenotypes</a>
<a href="#">cohortName</a>	Name of the cohort. For 'user-defined' this field could be generated upon the query, e.g. a value that is a concatenation or some representation of the user query.	string	NA
<a href="#">cohortSize</a>	Count of unique Individuals in cohort (individuals meeting criteria for 'user-defined' cohorts). If not previously known, it could be calculated by counting the individuals in the cohort.	integer	NA
<a href="#">cohortType</a>		string	NA



Field	description	type	properties
	Cohort type by its definition. If a cohort is declared 'study-defined' or 'beacon-defined' criteria are to be entered in cohort_inclusion_criteria; if a cohort is declared 'user-defined' cohort_inclusion_criteria could be automatically populated from the parameters used to perform the query.		
<a href="#">collectionEvents</a>	TBD	array	<a href="#">eventAgeRange</a> , <a href="#">eventCases</a> , <a href="#">eventControls</a> , <a href="#">eventDataTypes</a> , <a href="#">eventTimeline</a>

## 7.4 Datasets

Field	description	type	properties	example	enum
<a href="#">createDateTime</a>	The time the dataset was created (ISO 8601 format)	string	NA	2012-07-29, 2017-01-17T20:33:40Z	NA
<a href="#">dataUseConditions</a>	Data use conditions ruling this dataset	object	<a href="#">duoDataUse</a>	NA	NA
<a href="#">description</a>	Description of the dataset	string	NA	This dataset provides examples of the actual data in this Beacon instance.	NA
<a href="#">externalUrl</a>	URL to an external system providing more dataset information (RFC 3986 format).	string	NA	<a href="#">example.org/wiki/Main_Page</a>	NA
<a href="#">id</a>	Unique identifier of the dataset	string	NA	ds01010101	NA
<a href="#">info</a>	Placeholder to allow the Beacon to return any additional information that is necessary or could be of interest in relation to the query or the entry returned. It is recommended to encapsulate additional informations in this attribute instead of directly adding attributes at the same level than the others in order to avoid collision in the names of attributes in future versions of the specification.	object	NA	NA	NA
<a href="#">name</a>	Name of the dataset	string	NA	Dataset with synthetic data	NA
<a href="#">updateDateTime</a>	The time the dataset was updated in (ISO 8601 format)	string	NA	2012-07-19, 2017-01-17T20:33:40Z	NA
<a href="#">version</a>	Version of the dataset	string	NA	v1.1	NA

## 7.5 Genomic Variations

---

Field	description	type	properties
<a href="#">alternateBases</a>	Alternate bases for this variant (starting from <code>start</code> ). * Accepted values: IUPAC codes for nucleotides (e.g. <code>https://www.bioinformatics.org/sms/iupac.html</code> ). N is a wildcard, that denotes the position of any base, and can be used as a standalone base of any type or within a partially known sequence. As example, a query of <code>ANNT</code> the Ns can take any form of <code>[ACGT]</code> and will match <code>ANNT</code> , <code>ACNT</code> , <code>ACCT</code> , <code>ACGT</code> ... and so forth. <i>an empty value is used in the case of deletions with the maximally trimmed, deleted sequence being indicated in <a href="#">ReferenceBases</a></i> Categorical variant queries, e.g. such <i>not</i> being represented through sequence & position, make use of the <code>variantType</code> parameter. * Either <code>alternateBases</code> or <code>variantType</code> is required.	string	NA
<a href="#">caseLevelData</a>	<code>caseLevelData</code> reports about the variation instances observed in individual analyses.	array	<a href="#">alleleOrigin</a> , <a href="#">analysisId</a> , <a href="#">biosampleId</a> , <a href="#">clinicalInterpretations</a> , <a href="#">zygosity</a>
<a href="#">frequencyInPopulations</a>	NA	array	<a href="#">frequencies</a> , <a href="#">source</a> , <a href="#">sourceReference</a> , <a href="#">version</a>
<a href="#">identifiers</a>	NA	object	<a href="#">clinVarIds</a> , <a href="#">genomicHGVSId</a> , <a href="#">proteinHGVSIds</a> , <a href="#">transcriptHGVS</a>
<a href="#">molecularAttributes</a>	NA	object	<a href="#">aminoacidChanges</a> , <a href="#">geneIds</a> , <a href="#">genomicFeatures</a> , <a href="#">molecularEffects</a>
<a href="#">position</a>	This section groups all attributes that allows to identify a variant via its position in the genome.	object	<a href="#">assemblyId</a> , <a href="#">end</a> , <a href="#">refseqId</a> , <a href="#">start</a>
<a href="#">referenceBases</a>	Reference bases for this variant (starting from <code>start</code> ). * Accepted values: IUPAC codes for nucleotides (e.g. <code>https://www.bioinformatics.org/sms/iupac.html</code> ). N is a wildcard, that denotes the position of any base, and can be used as a standalone base of any type or within a partially known sequence. As example, a query of <code>ANNT</code> the Ns can take any form of <code>[ACGT]</code> and will match <code>ANNT</code> , <code>ACNT</code> , <code>ACCT</code> ,	string	NA

Field	description	type	properties
	<p>ACGT ... and so forth. * an <i>empty value</i> is used in the case of insertions with the maximally trimmed, inserted sequence being indicated in <code>AlternateBases</code>. NOTE: Many Beacon instances could not support UIPAC codes and it is not mandatory for them to do so. In such cases the use of [ACGTN] is mandated.</p>		
<a href="#">variantInternalId</a>	<p>Reference to the <b>internal</b> variant ID. This represents the primary key/identifier of that variant <b>inside</b> a given Beacon instance. Different Beacon instances may use identical id values, referring to unrelated variants. Public identifiers such as the GA4GH Variant Representation Id (VRSid) MUST be returned in the <code>identifiers</code> section. A Beacon instance can, of course, use the VRSid as their own internal id but still MUST represent this then in the <code>identifiers</code> section.</p>	string	NA
<a href="#">variantLevelData</a>	NA	object	<a href="#">clinicalInterpretations</a> , <a href="#">phenotypicEffects</a>
<a href="#">variantType</a>	<p>The <code>variantType</code> declares the nature of the variation in relation to a reference. In a response, it is used to describe the variation. In a request, it is used to declare the type of event the Beacon client is looking for. If in queries variants can not be defined through a sequence of one or more bases ( <code>precise</code> variants) it can be used standalone (i.e. without <code>alternateBases</code> ) together with positional parameters. Examples here are e.g. queries for structural variants such as <code>DUP</code> (increased allelic count of material from the genomic region between <code>start</code> and <code>end</code> positions without assumption about the placement of the additional sequence) or <code>DEL</code> (deletion of sequence following <code>start</code> ). Either <code>alternateBases</code> or <code>variantType</code> is required, with</p>	string	NA

Field	description	type	properties
	the exception of range queries (single <code>start</code> and <code>end</code> parameters).		

---

## 7.6 Individuals

---

Field	description	type	properties
diseases	List of disease(s) been diagnosed to the individual, defined by disease ontology ID(s), age of onset, stage and the presence of family history.	array	ageOfOnset, diseaseCode, familyHistory, notes, severityLevel, stage
ethnicity	Definition of an ontology term.	object	id, label
exposures	NA	array	ageAtExposure, date, duration, exposureCode, units, value
geographicOrigin	Definition of an ontology term.	object	id, label
id	Individual identifier (internal ID).	string	NA
info	Placeholder to allow the Beacon to return any additional information that is necessary or could be of interest in relation to the query or the entry returned. It is recommended to encapsulate additional informations in this attribute instead of directly adding attributes at the same level than the others in order to avoid collision in	object	NA



Field	description	type	properties
	the names of attributes in future versions of the specification.		
<a href="#">interventionsOrProcedures</a>	NA	array	<a href="#">ageAtProcedure</a> , <a href="#">bodySite</a> , <a href="#">dateOfProcedure</a> , <a href="#">procedureCode</a>
<a href="#">measures</a>	NA	array	<a href="#">assayCode</a> , <a href="#">date</a> , <a href="#">measurementValue</a> , <a href="#">notes</a> , <a href="#">observationMoment</a> , <a href="#">procedu</a>
<a href="#">pedigrees</a>	NA	array	<a href="#">disease</a> , <a href="#">id</a> , <a href="#">members</a> , <a href="#">numSubjects</a>
<a href="#">phenotypicFeatures</a>	NA	array	<a href="#">evidence</a> , <a href="#">excluded</a> , <a href="#">featureType</a> , <a href="#">modifiers</a> , <a href="#">notes</a> , <a href="#">onset</a> , <a href="#">resolution</a> , <a href="#">severityLevel</a>
<a href="#">sex</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>
<a href="#">treatments</a>	NA	array	<a href="#">ageAtOnset</a> , <a href="#">dose</a> , <a href="#">duration</a> , <a href="#">frequency</a> , <a href="#">route</a> , <a href="#">treatmentCode</a> , <a href="#">units</a>

## 7.7 Runs

---

Field	description	type	properties	example	enum
<a href="#">biosampleId</a>	Reference to the biosample ID.	string	NA	008dafdd-a3d1-4801-8c0a-8714e2b58e48	NA
<a href="#">id</a>	Run ID.	string	NA	SRR10903401	NA
<a href="#">individualId</a>	Reference to the individual ID.	string	NA	TCGA-AO-A0JJ	NA
<a href="#">info</a>	Placeholder to allow the Beacon to return any additional information that is necessary or could be of interest in relation to the query or the entry returned. It is recommended to encapsulate additional informations in this attribute instead of directly adding attributes at the same level than the others in order to avoid collision in the names of attributes in future versions of the specification.	object	NA	NA	NA
<a href="#">libraryLayout</a>	Ontology value for the library layout e.g "PAIRED", "SINGLE" #todo add Ontology name?	string	NA	NA	PAIRED, SINGLE
<a href="#">librarySelection</a>	Selection method for library preparation, e.g "RANDOM", "RT-PCR"	string	NA	RANDOM, RT-PCR	NA
<a href="#">librarySource</a>	Definition of an ontology term.	object	<a href="#">id</a> , <a href="#">label</a>	NA	NA
<a href="#">libraryStrategy</a>	Library strategy, e.g. "WIG'S"	string	NA	NA	NA
<a href="#">platform</a>	General platform technology label. It SHOULD be a subset of the platformModel and used only for	string	NA	Illumina, Oxford Nanopore, Affymetrix	NA

Field	description	type	properties	example	enum
	query convenience, e.g. "return everything sequenced with Illumina", where the specific model is not relevant				
<a href="#">platformModel</a>	Definition of an ontology term.	object	<a href="#">id, label</a>	NA	NA
<a href="#">runDate</a>	Date at which the experiment was performed.	string	NA	NA	NA