Beacon v2 Documentation

None

Beacon API Development Team

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 Stakeholder 5
3. TODO, Bugs & Changes 6
   3.1 TODO 6
   3.2 Changes 6
4. Beacon Types 9
   4.1 Beacon v2 9
   4.2 Beacon "Flavours" 10
5. Implement & Deploy a Beacon 11
   5.1 Which are the implementation options? 11
   5.2 Are you planning to implement a Beacon v2 API from scratch? 14
   5.3 Formats, Standards and Integrations 16
   5.4 Beacon Implementations 18
   5.5 Beacon Networks 19
6. Beacon Components 20
   6.1 Beacon v2 Framework 20
   6.2 Models 24
   6.3 Filters 48
   6.4 Beacon REST API 51
7. Example Queries 52
   7.1 Genomic Variant Queries 52
8. Citation(s) 57
   8.1 Beacon v2 57

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 patterns (wildcards) 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.

User

- 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).

Deployer

- 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

Implementer

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

2.3 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

- [ ] ontent for Standards Integration (as noted on page)
- [ ] e-structuring of Documentation Scopes
- [ ] e-structuring of Framework page
- [x] e-structuring of Models page
    - integration with the Schemas pages
    - removal of the individual schemas from the main navigation
- [ ] elete documentation in framework and model repo READMEs and point here
- [ ] x https (probably @mbaudis has to do some registrar configuration...)
- [x] ontent for the Filters page
- [ ] erging (?) of the Introduction and Documentation Scopes pages
    - clarification, re-structuring, links ...
- [ ] omplete Implementations
- [ ] xtend Query documentation, also using content from the variant scouts document
- [ ] clude a table comparing different "variant types" between definitions (e.g. like below)

Beacon	VCF	SO	EFO	VRS	Note about Beacon use
DUP	DUP	SO: 0001742	EFO: 0030070	–low-level gain –high-level gain	Increase of allele count compared to locally expected baseline w/o expectation about localization of added sequence copies
DEL	...	...	...	...	...

- [ ] reate content for Tips for Implementers

3.1.2 Repositories

- [x] ame and title change
    - adjusting all the links accordingly
- [x] UG: fix transfer of examples
- [ ] etiring of framework and model repos

3.2 Changes

2022-03-24: Retiring Separate *Implementations* Repository

Example implementations have been moved to from the `implementations-v2` repository to the Beacon v2 Documentation - web access here.

2022-03-23: Name Change to `beacon-v2`

The repository name and file paths have been changed from `beacon-v2-unity-testing` to `beacon-v2`. Added miscellanea suggestions from Jorge (not all of them).

2022-03-22: More Reorganization of Navigation

- Moved content from `implementations-and-networks` to `other-implementations` and left only the "Networks" Part.
- Added `mkdocs-mermaid-plugin` both to `mkdocs.yaml` and to github workflows.
- Moved Schemas (Markdown Tables) and Terms List from main navigation to `Beacon Compoments/Models`
- Reorganization of navigation
- Added pages: Tips for Implementers

2022-03-21: Reorganization of navigation

- Reorganization of navigation
- Deleted page `implement-and-deploy.md`
- Added pages: What is Beacon v2 and Implementation options
- Filters Page Updated

2022-03-18: 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/tree/main/models/src`
- this can be used inline as `{{ no such element: mkdocs.config.base.Config object['repo_model_url'] }}`

2022-03-16: Documentation Content and Formats Updates

- ✓ ddition of more variant query examples
- ✓ ew landing pages for Implementations and Networks and Standards Integration
- ✓ any 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 meintained in this repository. We're testing rendered versions (same text/code base) through Github actions (here) and ReadTheDocs.

- ✓ esting 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 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/main/framework/json/responses/sections/beaconInformationalResponseMeta.json

2022-02-23: Re-tool

- ✓eplacement of the previopus general `yamler.py` with a dedicated `beaconYamler.py`
- ✓oving replacements to bin/config.yaml
- ✓equirement for complete arguments (in and out paths, in- and out formats) - see bin/yamlerRunner.sh

2022-02-22: Creation of repository

- ✓esign of directory structure
- ✓est tooling & population with auto-converted files

4. Beacon Types

4.1 Beacon v2

Beacon v2 is a **protocol/specification** established by the **Global Alliance for Genomics and Health initiative** (GA4GH) that defines an open standard for federated discovery of genomic (and phenoclinic) data in biomedical research and clinical applications.

Disclaimer

Beacon v2 is in the process of being approved by the GA4GH. All information that you will find in this document is **provisional**.

The current version of the protocol is **v2** and consists of two components, the *Framework* and the *Models*.

The Framework contains the format for requests and responses, whereas the Models contain the data structure (schemas) for the biological data. The overall function of these components is to provide the instructions to design a **REST API** (REpresentational State Transfer Application Programming Interface) with **OpenAPI** Specification (OAS). The OAS defines a standard, language-agnostic interface that is used by software developers to implement REST APIs.

Citation

Beacon v2 and Beacon Networks: a "lingua franca" for federated data discovery in biomedical genomics, and beyond. Jordi Rambla, Michael Baudis, Tim Beck, Lauren A. Fromont, Arcadi Navarro, Manuel Rueda, Gary Saunders, Babita Singh, J.Dylan Spalding, Juha Tornroos, Claudia Vasallo, Colin D.Veal, Anthony J.Brookes. *Human Mutation* (2022) DOI.

4.2 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[1] 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.

4.2.1 Boolean Response Beacon

A *Boolean* Beacon is in it's 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

TBD

Beacon Count Response

TBD

4.2.2 Beacons Supporting Data and Information Delivery

TBD

Beacon Data Model

TBD

Alternative Data Models

TBD

Data Handover

TBD

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

5. Implement & Deploy a Beacon

Important

As previously described, **Beacon v2 is an specification** for sharing/discovery of data. Thus, *a priori*, it has nothing to do with any particular software, database or computer language.

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.

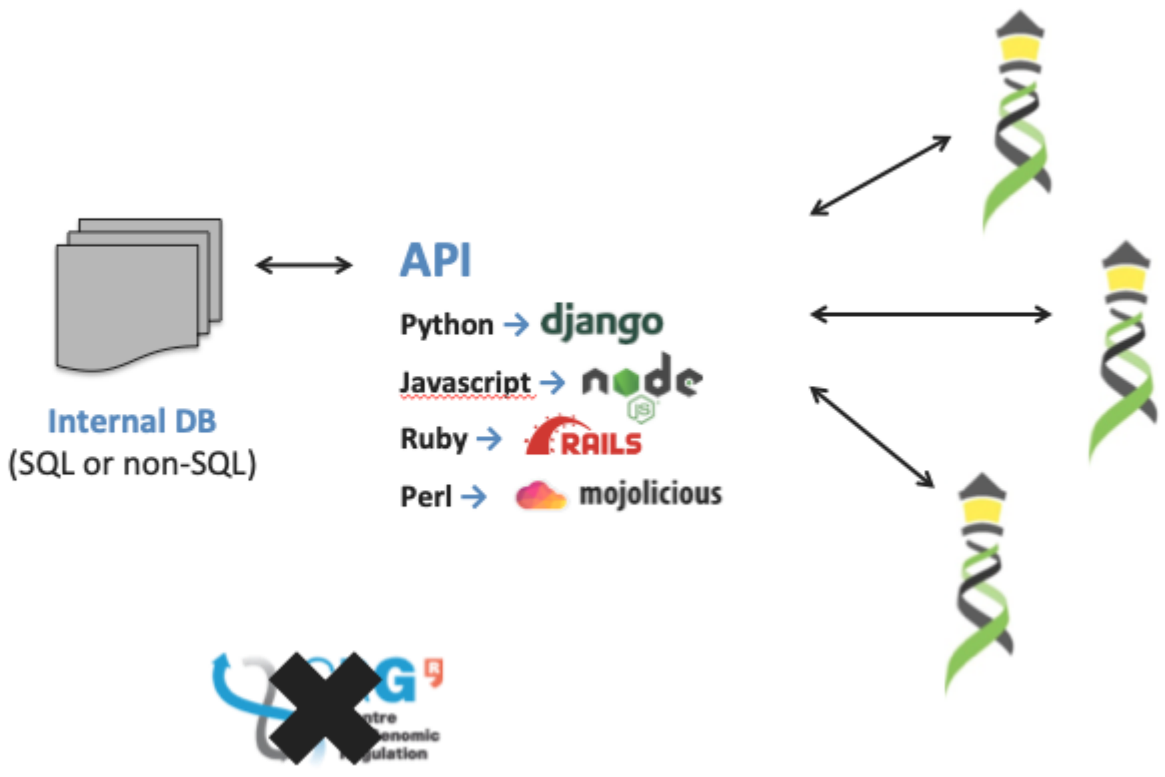
5.1 Which are the implementation options?

Two elements are needed to implement (or "light") a Beacon v2:

1. An internal **database** (where the biological data are stored).
2. A **REST API** that provides a standardized way to send queries and receive responses (containing yes/no, counts or data).

In this section we are going to present three implementation options, going from no involvement/delegate to CRG software to full delegate to CRG software.

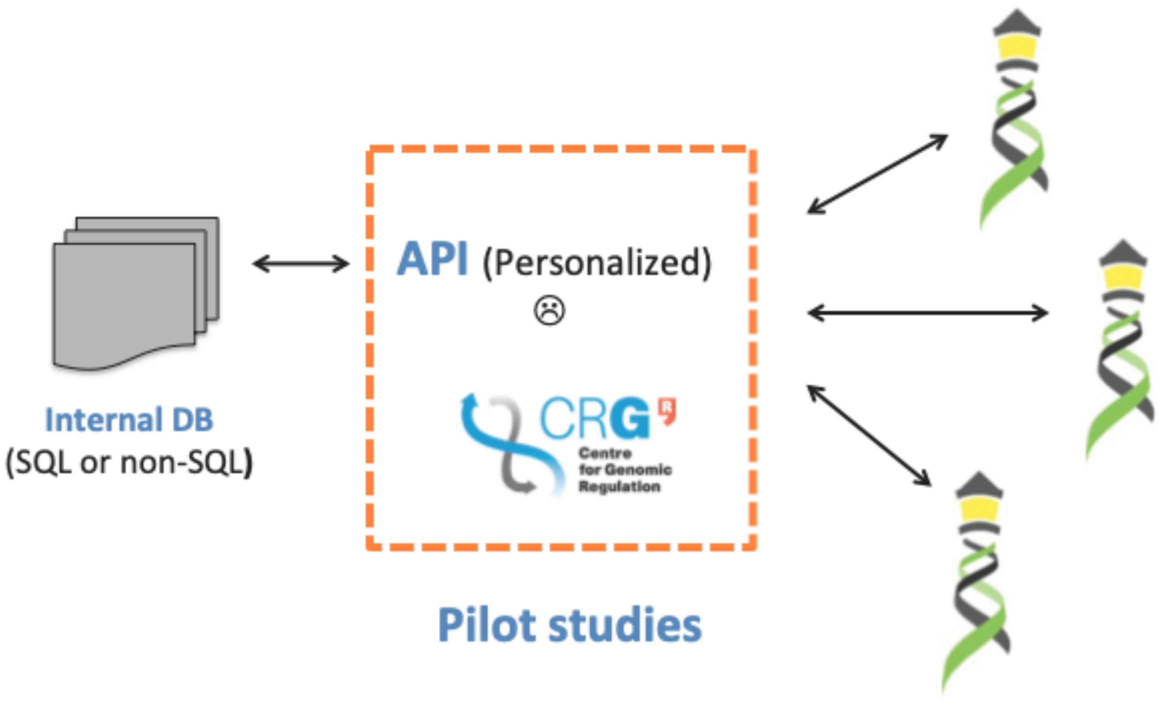
5.1.1 Option A



Let's say that you have your data organized and structured in a database (e.g. SQL or NoSQL which may or may not have an internal layer to get access to it). Let's also say that you have the resources (and knowledge) to read the "instructions" (i.e., Beacon v2 specification) to build an API on top of your existing solution. If that's your case, then this is the option for you. You are one of what we call **Beacon v2 API implementers**. We have a few of them already in the Beacon v2 Service Registry:

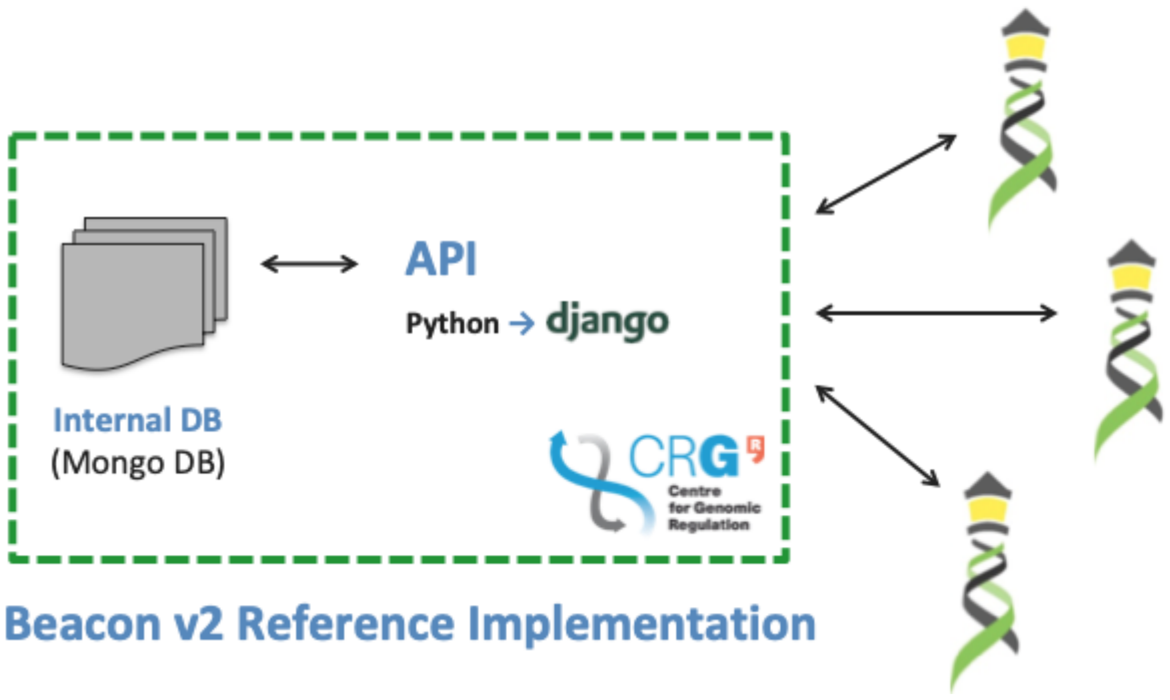
- European Genome-Phenome Archive Beacon
- Cafe Variome
- Progenetix Beacon+
- Fundacion Progreso y Salud Beacon v2 API
- CNAG Beacon v2 API

5.1.2 Option B



Let's say that you have a solution to organize your data but you don't have the resources (or knowledge) to implement a Beacon v2 API yourself. In some pilot studies, CRG has been helping individual institutions to build their Beacon v2 API. However, this option is not practical and does not scalate well so you may want to check **Option C**.

5.1.3 Option C


API
Python → django
Internal DB
(Mongo DB)
CRG Centre for Genomic Regulation
Beacon v2 Reference Implementation

Let's say that you have your data *somewhat* structured (you may have Excel files, PDFs, VCFs... or maybe a SQL database, or an EHR solution with phenoclinic information).

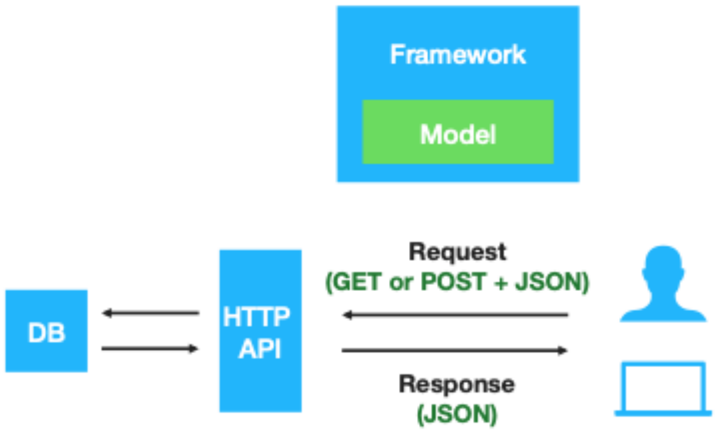
You want to "beaconize" your data to be part of a larger ecosystem, but you're unsure where to start, and/or don't want to invest a lot of resources because you are still unsure if the whole thing will pay off. Well, you're a not alone! Most centers are in this situation. For that reason at CRG we developed the **Beacon v2 Reference Implementation**.

Important

People that download and install *B2RI* are named **Beacon v2 deployers**.

5.2 Are you planning to implement a Beacon v2 API from scratch?

If this is the case, these are some things to get you started:



1. Start with a boolean beacon
2. List your available endpoints
3. Map the data from your DB to the Models
4. Handle the different types of filters
5. Build a response following the Framework

5.2.1 Request Example

```
{
  "meta": {
    "apiVersion": "v2.0",
    "requestedSchemas": [
      {
        "entityType": "EntryTypeA",
        "schema": "entry-typeA-schema-v2.0"
      }
    ]
  },
  "query": {
    "requestParameters": {
      "datasets": {
        "datasetIds": [
          "DatasetXYZ",
          "Dataset123"
        ]
      }
    },
    "filters": [
      {
        "id": "EFO:0001212",
        "scope": "biosamples"
      }
    ],
    "includeResultsetResponses": "HIT",
    "pagination": {
      "skip": 0,
      "limit": 10
    },
    "requestedGranularity": "count"
  }
}
```

5.2.2 Response Example

```
{
  "meta": {
    "beaconId": "org.example.beacon.v2",
    "apiVersion": "v2.0",
    "returnedSchemas": [
      {
        "entityType": "EntryTypeA",
        "schema": "entry-typeA-schema-v2.0"
      }
    ],
    "receivedRequestSummary": {
      "apiVersion": "v2.0",
      "filters": "...",
      "requestParameters": "...",
      "includeResultsetResponses": "HIT",
      "requestedSchemas": [
        {
          "entityType": "EntryTypeA",
          "schema": "entry-typeA-schema-v2.0"
        }
      ],
      "pagination": {
        "skip": 0,
        "limit": 10
      },
      "requestedGranularity": "count"
    }
  },
  "responseSummary": {
    "exists": true,
    "numTotalResults": 25355
  },
  "info": {
    "someInterestingStuff": "this is the interesting stuff"
  },
  "beaconHandovers": [
    {
      "handoverType": {
        "id": "EFO:0004157",
        "label": "BAM format"
      },
      "url": "https://api.mygenomeservice.org/Handover/9dcc48d7-fc88-11e8-9110-b0c592dbf8c0",
      "note": "This handover link provides access to a summarized VCF."
    }
  ]
}
```

5.3 Formats, Standards and Integrations

5.3.1 Data Formats and Standards

Coding and naming conventions

For historical reasons, in the names of entities, parameters and URLs we are following these 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.

Schema Language and Conventions

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

Genome Coordinates

0-based interbase etc., see schemablocks.org

Dates and Times

Date and time formats are specified as ISO8601 compatible strings, both for time points as well as for durations. Some of the ISO8601 compatible formats have not (yet) been used in the Beacon v2 default model.

PUBLIC SERVICE ANNOUNCEMENT:
OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.
THIS IS THE CORRECT WAY TO WRITE NUMERIC DATES:
2013-02-27
THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:
02/27/2013 02/27/13 27/02/2013 27/02/13
20130227 2013.02.27 27.02.13 27-02-13
27.2.13 2013. II. 27. 27/2-13 2013.158904109
MMXIII-II-XXVII MMXIII LVII/CCCLXV 1330300800
((3+3)×(111+1)-1)×3/3-1/3³ 2013 HISSSS 2-27-13
10/11011/1101 02/27/20/13 01237

EXAMPLES

- time stamp in milliseconds in YYYY-MM-DDTHH:MM:SS.SSS
  - *2015-02-10T00:03:42.123Z*

    –schema specification in JSON Schema is `"type": "string", format": "date-time"`

    –Timepoints with millisecond granularity are typical use cases for timing computer generated entries, e.g. the time of a record's update ("updateTime").

- age in years and months in PnYnM
  - *P43Y08M*

LINK: W3C **ISO8601**

LINK: ISO8601 documentation at **GA4GH SchemaBlocks**

5.3.2 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).

Variant Representation Standard (VRS)

(TBD)

LINK: **VRS Documentation**

Phenopackets

(TBD)

LINK: **Phenopackets Documentation**

5.4 Beacon Implementations

5.4.1 Others

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

Example Implementations

PROGENETIX API

The Progenetix database and cancer genomic information resource contains genome profiles of more than 140000 individual cancer genome screening experiments, with the majority representing results from genomic copy number assessment studies. With its Beacon+ 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

CAFE VARIOME

Link: Cafe Variome

CNAG BEACON V2 API

Link: CNAG Beacon v2 API

FUNDACION PROGRESO Y SALUD BEACON V2 API

Link: Fundacion Progreso y Salud Beacon v2 API

5.5 Beacon Networks

(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 nanimg 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 taylored 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*, ând *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 transerval 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 implentation 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
`boolean`	returns 'true/false' responses.
`count`	adds the total number of positive results found.
`aggregated`	returns summary, aggregated or distribution like responses per collection.
`record`	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
`PUBLIC`	Any anonymous user can read the data
`REGISTERED`	Only known users can read the data
`CONTROLLED`	Only specificly 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 Models

6.2.1 beacon-v2-Models

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 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 Framewrok 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.

```
classDiagram

    cohorts <-- genomicVariations : 1..n
    datasets <-- genomicVariations : 1..n
    cohorts <-- individuals : 1..n
    datasets <-- individuals : 1..n
    individuals <-- biosamples : 1..n
    biosamples <-- analyses : 1..n
    biosamples <-- runs : 1..n
    runs <-- genomicVariations : 1..n
    analyses <-- genomicVariations : 1..n
    genomicVariations <-- datasets : 1..n
    genomicVariations <-- cohorts : 1..n

    class biosamples{
        biosampleStatus
        collectionDate
        ...
    }
    class individuals{
        diseases
        ethnicity
        ...
    }
    class datasets{
        createDateTime
        dataUseCondition
        ...
    }
    class runs{
        biosampleId
        Id
        ...
        }
    class genomicVariations{
        alternateBases
        caseLevelData
        ...
    }
    class analyses{
        aligner
        analysisDate
        ...
    }
   class cohorts{
        cohortDataTypes
        cohortDesign
        ...
    }
```

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.

6.2.2 Default Schemas

Analyses

Field	Description	Type	Properties	Example	Enum
aligner	Reference to mapping/alignment software	string	NA	bwa-0.7.8	NA
analysisDate	Date at which analysis was performed.	string	NA	NA	NA
biosampleId	Reference to Biosample ID.	string	NA	S0001	NA
id	Analysis reference ID (external accession or internal ID)	string	NA	NA	NA
individualId	Reference to Individual ID.	string	NA	P0001	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 the names of attributes in future versions of the specification.	object	NA	NA	NA
pipelineName	Analysis pipeline and version if a standardized pipeline was used	string	NA	NA	NA
pipelineRef	Link to Analysis pipeline resource	string	NA	NA	NA
runId	Run identifier (external accession or internal ID).	string	NA	SRR10903401	NA
variantCaller	Reference to variant calling software / pipeline	string	NA	GATK4.0	NA

Biosamples

Field	description	type	properties
biosampleStatus	Definition of an ontology term.	object	id, label
collectionDate	Date of biosample collection in ISO8601 format.	string	NA
collectionMoment	Individual's or cell cullture 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
diagnosticMarkers	Clinically relevant biomarkers. RECOMMENDED.	array	id, label
histologicalDiagnosis	Definition of an ontology term.	object	id, label
id	Biosample identifier (external accession or internal ID).	string	NA
individualId	Reference to the individual from which that sample was obtained.	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 the names of attributes in future versions of the specification.	object	NA
measurements	List of measurements of the sample.	array	assayCode, date, measurementValue, notes, observation
notes	Any relevant info about the biosample that does not fit into any other field in the schema.	string	NA
obtentionProcedure	Class describing a clinical procedure or intervention.	object	ageAtProcedure, bodySite, dateOfProcedure, procedure
pathologicalStage	Definition of an ontology term.	object	id, label
pathologicalTnmFinding	Pathological TNM findings, if applicable, preferably as subclass of NCIT:C48698 - Cancer TNM Finding Category (NCIT:C48698). RECOMMENDED.	array	id, label
phenotypicFeatures	List of phenotypic abnormalities of the sample. RECOMMENDED.	array	evidence, excluded, featureType, modifiers, notes, onset severityLevel
sampleOriginDetail	Definition of an ontology term.	object	id, label
sampleOriginType	Definition of an ontology term.	object	id, label

Field	description	type	properties
sampleProcessing	Definition of an ontology term.	object	id, label
sampleStorage	Definition of an ontology term.	object	id, label
tumorGrade	Definition of an ontology term.	object	id, label
tumorProgression	Definition of an ontology term.	object	id, label

Cohorts

Field	description	type	properties
cohortDataTypes	Type of information. Preferably values from Genomics Cohorts Knowledge Ontology (GeCKO) or others when GeCKO is not applicable.	array	id, label
cohortDesign	Definition of an ontology term.	object	id, label
cohortExclusionCriteria	Criteria used for defining the cohort. It is assumed that all cohort participants will match or NOT match such criteria.	object	ageRange, diseaseConditions, ethnicities, genders, locations, phe
cohortId	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
cohortInclusionCriteria	Criteria used for defining the cohort. It is assumed that all cohort participants will match or NOT match such criteria.	object	ageRange, diseaseConditions, ethnicities, genders, locations, phe
cohortName	Name of the cohort. For ´user-defined´ this field could be generated upon the query, e.g. a value that is a concatenationor some representation of the user query.	string	NA
cohortSize	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
cohortType		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.		
collectionEvents	TBD	array	eventAgeRange, eventCases, eventControls, eventDataTypes, eve eventTimeline

Datasets

Field	description	type	properties	example	enum
createDateTime	The time the dataset was created (ISO 8601 format)	string	NA	2012-07-29, 2017-01-17T20:33:40Z	NA
dataUseConditions	Data use conditions ruling this dataset	object	duoDataUse	NA	NA
description	Description of the dataset	string	NA	This dataset provides examples of the actual data in this Beacon instance.	NA
externalUrl	URL to an external system providing more dataset information (RFC 3986 format).	string	NA	example.org/wiki/Main_Page	NA
id	Unique identifier of the dataset	string	NA	ds01010101	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 the names of attributes in future versions of the specification.	object	NA	NA	NA
name	Name of the dataset	string	NA	Dataset with synthetic data	NA
updateDateTime	The time the dataset was updated in (ISO 8601 format)	string	NA	2012-07-19, 2017-01-17T20:33:40Z	NA
version	Version of the dataset	string	NA	v1.1	NA

Genomic Variations

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

Field	description	type	properties
	`ACGT` ... and so forth. * an *empty value* is used in the case of insertions with the maximally trimmed, inserted sequence being indicated in `AlternateBases`. 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.		
variantInternalId	Reference to the **internal** variant ID. This represents the primary key/identifier of that variant **inside** 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 `identifiers` section. A Beacon instance can, of course, use the VRSid as their own internal id but still MUST represent this then in the `identifiers` section.	string	NA
variantLevelData	NA	object	clinicalInterpretations, phenotypicEffects
variantType	The `variantType` 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 (`precise` variants) it can be used standalone (i.e. without `alternateBases`) together with positional parameters. Examples here are e.g. queries for structural variants such as `DUP` (increased allelic count of material from the genomic region between `start` and `end` positions without assumption about the placement of the additional sequence) or `DEL` (deletion of sequence following `start`). Either `alternateBases` or `variantType` is required, with	string	NA

Field	description	type	properties
	the exception of range queries (single `start` and `end` parameters).		

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.		
interventionsOrProcedures	NA	array	ageAtProcedure, bodySite, dateOfProcedure, procedureCode
measures	NA	array	assayCode, date, measurementValue, notes, observationMoment, procedu
pedigrees	NA	array	disease, id, members, numSubjects
phenotypicFeatures	NA	array	evidence, excluded, featureType, modifiers, notes, onset, resolution, severityLevel
sex	Definition of an ontology term.	object	id, label
treatments	NA	array	ageAtOnset, dose, duration, frequency, route, treatmentCode, units

Runs

Field	description	type	properties	example	enum
biosampleId	Reference to the biosample ID.	string	NA	008dafdd-a3d1-4801-8c0a-8714e2b58e48	NA
id	Run ID.	string	NA	SRR10903401	NA
individualId	Reference to the individual ID.	string	NA	TCGA-AO-A0JJ	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 the names of attributes in future versions of the specification.	object	NA	NA	NA
libraryLayout	Ontology value for the library layout e.g "PAIRED", "SINGLE" #todo add Ontology name?	string	NA	NA	PAIRED, SINGLE
librarySelection	Selection method for library preparation, e.g "RANDOM", "RT-PCR"	string	NA	RANDOM, RT-PCR	NA
librarySource	Definition of an ontology term.	object	id, label	NA	NA
libraryStrategy	Library strategy, e.g. "WIG'S"	string	NA	NA	NA
platform	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 Illimuna", where the specific model is not relevant				
platformModel	Definition of an ontology term.	object	id, label	NA	NA
runDate	Date at which the experiment was performed.	string	NA	NA	NA

6.2.3 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

6.3 Filters

Filters represent a powerful addition to the Beacon query API. They are rules for selecting records based upon the field values those records contain. The rules can refer to bio-ontology or custom terms, numerical or alphanumerical values, and employ wildcards, standard operators or other principles of selection. This empowers such options as queries for phenotypes, disease codes or technical parameters associated with observed genomic variants.

6.3.1 Filter types

A Beacon can support four types of Filters.

1. **Bio-ontology terms** for biomedical data or procedural metadata that are contained in public repositories such as the EMBL-EBI Ontology Lookup Service or the NCBO BioPortal. Bio-ontology terms are identified using the full term/class identifier as CURIE, e.g. “HP:0100526”.
2. **Custom terms** for biomedical or metadata terms that are locally defined by a Beacon (e.g. not corresponding to known bio-ontology terms). Custom terms must contain unique identifiers that are used in Beacon requests.
3. **Numerical values** include integer, decimal and float data types.
4. **Alphanumerical values** include alphabetic letters and special characters with or without numbers.

6.3.2 */filtering_terms* informational endpoint

The */filtering_terms* endpoint returns a list of all data fields whose values may be subjected to filtering, plus the data type(s) for those fields, and/or the list of extant values for each of those data fields in the current dataset. In addition, for each bio-ontology used by a Beacon, the endpoint response includes a description of the bio-ontology in Phenopackets Resource format.

The endpoint's `filterTerms` response identifies the Filter types.

Bio-ontology and custom term Filter types contain:

- `type` = resource name (required)
- `id` = term id (required)
- `label` = term label (optional)

```
"response":{
    "resources":[
        {
            "id":"hp",
            "name":"Human Phenotype Ontology",
            "url":"http://purl.obolibrary.org/obo/hp.owl",
            "version":"27-03-2020",
            "namespacePrefix":"HP",
            "iriPrefix":"http://purl.obolibrary.org/obo/HP_"
        },
        ...
    ],
    "filterTerms": [
        {
            "type": "Human Phenotype Ontology",
            "id": "HP:0008773",
            "label": "neoplasm of the lung"
        },
        ...
    ]
}
```

Numerical value Filter types contain:

- `type` = data type as 'numeric' (required)
- `id` = field id (required)
- `label` = field label (optional)

```
"filterTerms": [
    {
        "type": "numeric",
        "id": "PATO:000001",
        "label": "age"
    },
    ...
]
```

Alphanumerical value Filter types contain:

- `type` = data type as 'alphanumeric' (required)
- `id` = field id (required)
- `label` = field label (optional)

```
"filterTerms": [
    {
        "type": "alphanumeric",
        "id": "HP:0032443",
        "label": "past medical history"
    },
    ...
]
```

6.3.3 Using Filter queries

For all query types, the logical AND is implied between Filters. The Filter `id` is required for all query types.

Hierarchical ontology query

A Beacon will query for entities associated with the submitted bio-ontology term(s), and by default, all descendent terms. The optional `includeDescendantTerms` parameter can be set to either `true` or `false`. The default and assumed value of `includeDescendantTerms` is `true`, thus if the parameter is not set, then the use of bio-ontology terms in a Beacon request implies that a hierarchical ontology search is requested.

POST request example of three Filters, where one Filter excludes matches with descendent terms:

```
"filters": [
    {
        "id": "HP:0100526",
        "includeDescendantTerms": false
    },
    {
        "id": "HP:0005978"
    },
    {
        "id": "HP:0005978"
    }
]
```

Semantic similarity query

A Beacon will query for entities that are associated with bio-ontology terms that are similar to the submitted terms. The Beacon API is agnostic to the semantic similarity model implemented by a Beacon and how a Beacon applies the relative thresholds of similarity. A semantic similarity query request contains the required `similarity` parameter with a value set to define the relative threshold level of `high`, `medium` or `low`.

POST request example of two Filters using differing relative similarity thresholds:

```
"filters": [
    {
        "id": "HP:0100526",
        "similarity": "high"
    },
    {
        "id": "HP:0005978",
        "similarity": "medium"
    }
]
```

Numerical value query

A Beacon will query for quantitative properties when the required `operator` and numerical `value` parameters are set in the filters request. The `id` parameter identifies the field name, the `operator` parameter defines the operator to use, and the `value` parameter provides the field query value. Equality and relational operators (= < >) can be used between field name and field value pairs, and field values can be associated with units if applicable.

POST request example of a Filter for individuals over 70 years of age (age = PATO:0000011, age syntax as ISO 8601):

```
"filters": [
    {
        "id": "PATO:0000011",
        "operator": ">",
        "value": "P70Y"
    }
]
```

Alphanumerical value query

A Beacon will query free-text values within fields when the required `operator` and alphanumerical `value` parameters are set in the filters request. Queries can be for exact alphanumerical values, used to exclude alphanumerical values, or employ wildcards to match patterns within alphanumerical values. In all query classes, the `id` parameter identifies the field name, the `operator` parameter defines the operator to use, and the `value` parameter provides the field query value.

'EXACT' VALUE QUERY

The `operator` parameter is set to the equality (=) operator.

POST request example of using free-text to filter medical history (past medical history = HP:0032443):

```
"filters": [
    {
        "id": "HP:0032443",
        "operator": "=",
        "value": "unknown medical history"
    }
]
```

'LIKE' value query

The inclusion of a percent sign (%) wildcard character within the `value` parameter represents zero or more characters within a LIKE style string match. The wildcard character can lead the query string, end the string, or surround the string.

POST request example to filter medical history free-text for any reference to cancer:

```
"filters": [
    {
        "id": "HP:0032443",
        "operator": "=",
        "value": "%cancer%"
    }
]
```

'NOT' VALUE QUERY

The `operator` parameter is set to the logical not (!) operator. The `value` parameter should not be present in field value. The wildcard character can be used if required.

POST request example to filter medical history free-text for records that do not include the query string:

```
"filters": [
    {
        "id": "HP:0032443",
        "operator": "!",
        "value": "unknown medical history"
    }
]
```

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`

REST Endpoint Definitions

The endpoind paths available for a given Beacon instance are defined in `__APIroot__/beaconMap/` Github

Typical Endpoint Patterns

(TBD)

7. Example Queries

7.1 Genomic Variant Queries

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

7.1.1 Beacon *Sequence Queries*

Sequence Queries query for the existence of a specified sequence at a given genomic position. Such queries correspond to the original Beacon queries and are used to match short, precisely defined genomic variants such as SNVs and INDELs.

PARAMETERS

- `referenceName`
- `start` (single value)
- `alternateBases`
- `referenceBases`

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/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.

7.1.2 Beacon *Range Queries* and *GeneId Queries*

Beacon *Range Queries* are supposed to return matches of any variant with at least partial overlap of the sequence range specified by `reference_name` , `start` and `end` parameters.

GeneId Queries are in essence a variation of *Range Queries* in which the coordinates are replaced by the HGNC gene symbol. It is left to the implementation if the matching is done on variants annotated for the gene symbol or if a positional translation is being applied.

⚠ Use of `start` and `end`

Range queries require the use of **single** `start` and `end` parameters, in contrast to *Bracket Queries*.

PARAMETERS

- `referenceName`
- `start` (single value)
- `end` (single value)
- optional
  - `variantType` **OR** `alternateBases` **OR** `aminoacidChange`
  - `variantMinLength`
  - `variantMaxLength`

EXAMPLE: ANY VARIANT AFFECTING *EIF4A1*

Beacon v2 GET **Beacon v2 GET for `geneId`** **Beacon v2 POST** **Beacon v1** **Beacon v0.3**

```
?assemblyId=GRCh38&referenceName=17&start=7572837&end=7578641
```

```
?geneId=EIF4A1
```

```
{
    "$schema":"https://raw.githubusercontent.com/ga4gh-beacon/beacon-v2/main/framework/json/requests/beaconRequestBody.json",
    "meta": {
        "apiVersion": "2.0",
        "requestedSchemas": [
            {
                "entityType": "genomicVariation",
                "schema:": "https://raw.githubusercontent.com/ga4gh-beacon/beacon-v2/main/models/json/beacon-v2-default-model/genomicVariations/
defaultSchema.json"
            }
        ]
    },
    "query": {
        "requestParameters": {
            "referenceName": "NC_000017.11",
            "start": [ 7572837 ],
            "end": [ 7578641 ]
        }
    },
    "requestedGranularity": "record",
    "pagination": {
        "skip": 0,
        "limit": 5
    }
}
```

Range Queries are new to Beacon v2

Range Queries are new to Beacon v2

7.1.3 Beacon *Bracket Queries*

Bracket Queries allow the specification of sequence ranges for both start and end positions of a genomic variation. The typical example here is the query for similar structural variants - particularly CNVs - affecting a genomic region but potentially differing in their exact base extents.

Use of `start` and `end`

Bracket queries require the use of **two** `start` and `end` parameters, in contrast to *Range Queries*.

PARAMETERS

- `referenceName`
- `start` (min) and `start` (max) - i.e. 2 start parameters
- `end` (min) and `end` (max) - i.e. 2 end parameters
- `variantType` (optional)

EXAMPLE: CNV QUERY - *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 occurr 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/main/framework/json/requests/beaconRequestBody.json",
    "meta": {
        "apiVersion": "2.0",
        "requestedSchemas": [
            {
                "entityType": "genomicVariation",
                "schema:": "https://raw.githubusercontent.com/ga4gh-beacon/beacon-v2/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$^+$ prototyping.

7.1.4 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 vatiant with partial or complete overlap with this sequence range
  - additional parameters (e.g. `referenceBases` , `alternateBases` , `variantType` ...) may be used to scope the range query
- query by `aminoacidChange`

- query by `geneId`
- `variantMinLength`, `variantMaxLength`

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`

8. Citation(s)

8.1 Beacon v2

Citation

Beacon v2 and Beacon Networks: a "lingua franca" for federated data discovery in biomedical genomics, and beyond. Jordi Rambla, Michael Baudis, Tim Beck, Lauren A. Fromont, Arcadi Navarro, Manuel Rueda, Gary Saunders, Babita Singh, J.Dylan Spalding, Juha Tornroos, Claudia Vasallo, Colin D.Veal, Anthony J.Brookes. *Human Mutation* (2022) DOI.