Global Alliance
for Genomics & Health

GA4GH Schemas Documentation

Release 0.0.1

Jeltje van Baren

Sep 07, 2017

Contents

Global Alliance
for Genomics & Health

The GA4GH API attempts to gather together protocols and data models useful for genomics data interchange. It offers protocols that can be implemented over existing Genomics data stores to make these results more easily discovered, shared, and replicated.

CHAPTER 1

Contents

Introduction

The Data Working Group of the Global Alliance for Genomics and Health developed this web API to facilitate access to and exchange of genomics data across remote sites.

Why use this web API?

This API is specifically designed to allow sharing of genomics data in a standardized manner and without having to exchange complete experiments. With this API, you can

- Exchange genome annotations, DNA sequence reads, reference-based alignments, metadata, and variant calls
- Request alignments and variant calls for one genome or a million.
- Explore data by slicing alignments and variants by genomic location or other feature across one or multiple genomes.
- Interactively process entire cohorts

If you need background

For more details on web APIs, see this wikipedia page. ReST protocols for data transfer are described here. Wikipedia also has good overviews of bioinformatics and DNA sequencing

The GA4GH web API

This API was created to enable researchers to better access and exchange genomic data across remote sites. For example, instead of downloading complete BAM files or whole genome annotations, the API allows retrieval of information on, for instance, single genes or genomic regions.

For a full list of the GA4GH API goals, see *API Goals*

Schemas and formats

The API consists of a series of schemas that define the types of things that API clients and servers exchange: requests for data, server responses, error messages, and objects actually representing pieces of genomics data.

The schemas are written in Protocol Buffers Interface Description Language (extension .proto). For more details on Protocol Buffers and how it is used in the GA4GH APIs, see *Google Protocol Buffers*.

Here is an example schema definition for a Variant (with comments removed):

```
message Variant {
  string id = 1;
  string variant_set_id = 2;
  repeated string names = 3;
  int64 created = 4;
  int64 updated = 5;
  string reference_name = 6;
  int64 start = 7;
  int64 end = 8;
  string reference_bases = 9;
  repeated string alternate_bases = 10;
  map<string, google.protobuf.ListValue> info = 11;
  repeated Call calls = 12;
}
```

On the wire, the GA4GH web API takes the form of a client and a server exchanging JSON-serialized objects over HTTP or HTTPS. For more details on JSON, including how the GA4GH web API serializes and deserializes Protocol Buffers objects in JSON, see *The JSON Format*.

API Overview

The API is designed for sharing genomic data. It currently has support for sharing sequencing reads, genetic variants, and reference genomes. The following sections give an overview of the API, including general API patterns, individual data types, and links to detailed schema documentation.

API Design

Object Ids

- Many objects need to be referenced by the API (e.g. a search method may return a list of objects) and by applications built outside the API (e.g. a visualization app may refer back to the objects being shown). The API has a standard mechanism for referring to such objects.
- The standard way to programmatically reference an object is via an **id** field. The id is server-assigned, and must provide "id-like" semantics, including:
  - ids are unique within the scope of the server instance
  - ids are durable for the lifetime of the server, and persistent across server restarts – once a user is given an ID for data stored on a server, the id remains valid for as long as the server is still storing that data
- Many objects, including most 'container' objects, also have a **name** field. The name is user-defined and is intended to be human readable. This can be thought of as a display name.
- Reference names within a Reference Set are expected to be unique. There are no other uniqueness requirements on names.

Cross-repository data federation will need a standard way to refer to a data object, regardless of which repository it's in. There is no such standard currently, and the current API, including the id and name fields, isn't sufficient. A future API may introduce standard cross-repository identifiers using some combination of **content hashes**, **GUIDs**, and central **accession** facilities.

ID and Name

Throughout the API objects have *IDs*. The purpose of IDs is to allow unique identification of all objects within a single server, such that no two objects in a given server have the same ID and no object has more than one ID. The scope of an ID is limited to a given server and an ID may be an arbitrary string.

A name is a user defined identifier. Names need only be uniquely identifying within a specific scope, for example, the names of sequences within a ReferenceSet must be distinct, but there might be two sequences named "chr1" stored in a server, each in a different ReferenceSet. Names may be an arbitrary string.

Object Relationships

- Some objects are contained by other objects. These relationships can be
  - many:1 (e.g. ReadGroupSets in a Dataset); aka single-include
  - many:many (e.g. ReadGroups in a ReadGroupSet); aka multi-include
- Some objects are derived from other objects. These relationships can be
  - many:1 (e.g different aligned ReadGroupSet's derived from an unaligned ReadGroupSet using different alignment algorithms and/or reference sequences)
  - many:many (e.g. different VariantSets derived from a collection of ReadGroupSets using different joint variant calling algorithms)

Dataset

A dataset is a highest level grouping that contains sequence and variant data. It provides the concept of a container which allows high level separation between data.

For the Dataset schema definition see the Metadata schema

Unresolved Issues

- Is the GA4GH object design a conceptual data model that must be followed or only containers for data exchange. If they are containers, where is the conceptual data model defined?
- Are GA4GH objects idempotent? In particular, can one obtain an object with a subset of it's fields?
- Is object life-cycle semantics in the scope of GA4GH API? Which objects are immutable and which are mutable? If objects are mutable, how does one know they have changed? How does one protect against changes while using the objects over a given time-frame?
- What is the definition of the wire protocol? HTTP 1.0? Is HTTP 1.1 chunked encoding allowed? What is the specification for the generated JSON for a given an Protocol Buffers schema?
- What is the role of Protocol Buffers? Is it for documentation-only or for use as an IDL?
- Need overall object relationship diagram.

API Goals

- From the GA4GH DWG site:

  > The Global Alliance for Genomics and Health (GA4GH) Genomics [API] will allow the interoperable exchange of genomic information across multiple organizations and on multiple platforms. This is a freely available open standard for interoperability, that uses common web protocols to support serving and sharing of data on DNA sequences and genomic variation. The API is implemented as a Web service to create a data source which may be integrated into visualization software, web-based genomics portals or processed as part of genomic analysis pipelines. It overcomes the barriers of incompatible infrastructure between organizations and institutions to enable DNA data providers and consumers to better share genomic data and work together on a global scale, advancing genome research and clinical application.

- The API must allow flexibility in server implementation, including:
  - choice of persistent backend (e.g. files, SQL, NoSQL)
  - choice of implementation language (e.g. Java, Python, Go)
  - choice of authorization model (e.g. all public, all private, fine-grain ACLs)
  - choice of import mechanism (e.g. self-service vs. centrally managed)
  - choice of commercial model (e.g. single payer vs. per-data-owner billing)
  - choice of scale (from a single researcher working with dozens of sequences and homogeneous tools, to government-funded studies with over a million sequences and multiple tool chains)
- The API must allow full-fidelity representation of data that was prepared using today's common methods and stored using today's common file formats.
  - Note that real-world data files sometimes use invalid or ambiguous syntax, making it hard to understand the semantics of the contained data. If a server can't figure out what those semantics are, it can throw an error on import. But whenever the semantics are clear, including when they're specified in valid data files, the API must allow preserving them.
- The API should allow adding more structure to data beyond today's common practices (e.g. formal provenance, versioning).
- The API should allow data owners to organize their data in ways that make sense to them, which implies there often isn't One True Taxonomy. For example, a ReadGroupSet is defined as "a set of ReadGroups that are intended to be analyzed together" – different researchers might choose different sets for different purposes.
  - At the same time, the API should encourage reusable organization, anticipating a future that supports cross-researcher and cross-repository data federation.

Unresolved Issues

- What is the operational scope of the API? Is the capacity goal for a transfer? Should it handle small amounts of data randoman or larger transfers?
- How is sharing defined? Download of data for use in another environment or online, random access to the data?
- What are the performance goals of the API in various configurations?
- What is the scope of interoperability? Code-only interoperability or data interoperability?
- Is the DWG defining an API or a federated network of servers, tools to build a federated network of servers? Need to define the scope.
- Need high-level uses cases for entire API.

Reads

Reads are genetic data generated by a DNA sequencing instrument, including nucleotides and quality scores. Reads may optionally be aligned to a reference sequence. (The data model for reads is similar to SAM/BAM.)

Reads API

See Reads schema for a detailed reference.

Reads Data Model

The Reads data model,although based on the SAM format, allows for more versatile interaction with the data. Instead of sending whole chromosome or whole genome files, the server can send information on specific genomic regions instead.

The model has the following data types:

Record	Description	SAM/BAM rough equivalent
ReadAlignment	One alignment for one read	A single line in a file
ReadGroup	A group of read alignments	A single RG tag
ReadGroupSet	Collection of ReadGroups that map to the same genome	Single SAM/BAM file
Program	Software version and parameters that were used to align reads to the genome	PN, CL tags in SAM header
ReadStats	Counts of aligned and unaligned reads for a ReadGroup or ReadGroupSet	Samtools flagstats on a file

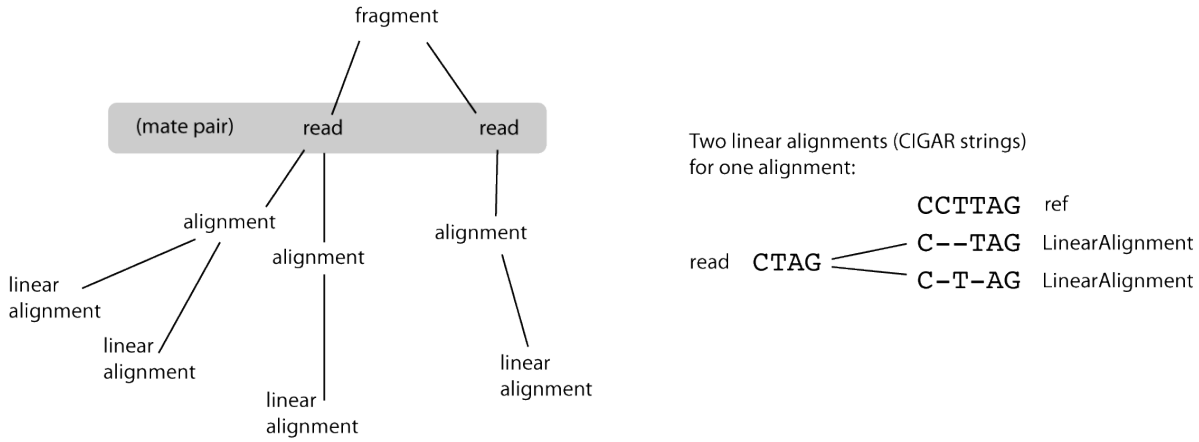
The relationships are mostly one to many (e.g. each *ReadAlignment* is part of exactly one *ReadGroup*), with the exception that a *ReadGroup* is allowed to be part of more than one *ReadGroupSet*.

Dataset –≺ *ReadGroupSet* ≻–≺ *ReadGroup* –≺ *ReadAlignment*

- A *Dataset* is a general-purpose container, defined in metadata.avdl.
- A *ReadGroupSet* is a logical collection of ReadGroups, as determined by the data owner. Typically one *ReadGroupSet* represents all the Reads from one experimental sample, which traditionally would be stored in a single BAM file.
- A *ReadGroup* is all the data that's processed the same way by the sequencer. There are typically 1-10 ReadGroups in a *ReadGroupSet*.
- A *ReadAlignment* object is a flattened representation of several layers of bioinformatics hierarchy, including fragments, reads, and alignments, stored in one object for easy access.

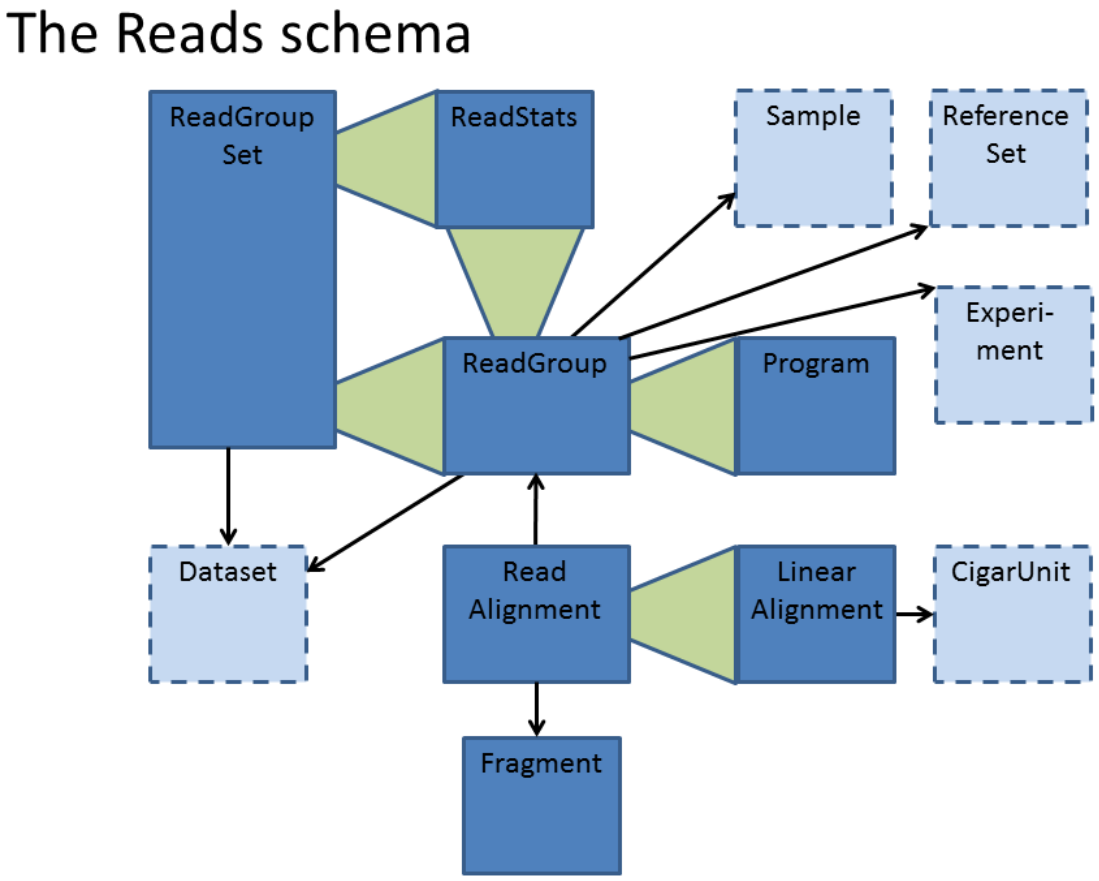
ReadAlignment: detailed discussion

One *ReadAlignment* object represents the following logical hierarchy. See the field definitions in the *ReadAlignment* object for more details.



- A *fragment* is a single stretch of a DNA molecule. There are typically at least millions of fragments in a ReadGroup. A fragment has a name (QNAME in BAM spec), a length (TLEN in BAM spec), and one or more reads.
- A *read* is a contiguous sequence of bases. There are typically only one or two reads in a fragment. If there are two reads, they're known as a mate pair. A read has an array of base values, an array of base qualities, and optional alignment information.
- An *alignment* is the way alignment software maps a read to a reference. There's one primary alignment, and can be one or more secondary alignments. Secondary alignments represent alternate possible mappings.
- A *linear alignment* maps a string of bases to a reference using a single CIGAR string. There's one representative alignment, and can be one or more supplementary alignments. Supplementary alignments represent linear alignments that are subsets of a chimeric alignment.

The image below shows which Reads records contain other records (represented by green triangles), and which contain IDs that can be used to get information from other records (arrows). The arrow points *from* the record that lists the ID *to* the record that can be identified by that ID. Records are represented by blue rectangles; dotted lines indicate records defined in other schemas.



Variants

Variants are genetic differences between an experimental sample and a reference sequence. (The data model for variants is similar to VCF.)

Variants API

See Variants schema for a detailed reference.

Variants Data Model

The Variants data model, although based on the VCF format, allows for more versatile interaction with the data. Instead of sending whole VCF files, the server can send information on specific variants or genomic regions instead. And instead of getting the whole genotype matrix, it's possible to just get details for one or more specified individuals.

The API uses four main entities to represent variants. The following diagram illustrates how these entities relate to each other to constitute the genotype matrix.

The lowest-level entity is a Call:

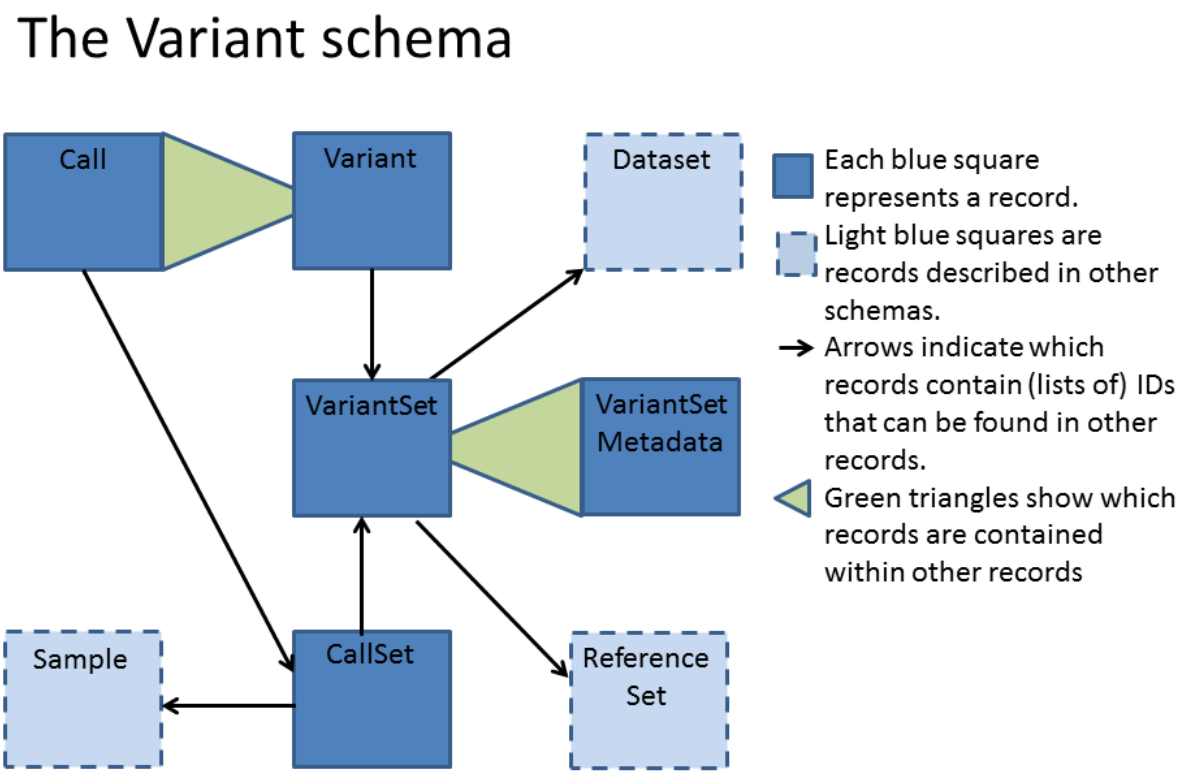
- a *Call* encodes the genotype of an individual with respect to a variant, as determined by some analysis of experimental data.

The other entities can be thought of as collections of Calls that have something in common:

- a *VariantSet* supports working with a collection of Calls intended to be analyzed together.
- a *Variant* supports working with the subset of Calls in a VariantSet that are at the same site and are described using the same set of alleles. The Variant entity contains:
  - a variant description: a potential difference between experimental DNA and a reference sequence, including the site (position of the difference) and alleles (how the bases differ)
  - variant observations: a collection of Calls describing evidence for actual instances of that difference, as seen in analyses of experimental data
- a *CallSet* supports working with the subset of Calls in a VariantSet that were generated by the same analysis of the same sample. The CallSet includes information about which sample was analyzed and how it was analyzed, and is linked to information about what differences were found.

The following diagram shows the relationship of these four entities to each other and to other GA4GH API entities. It shows which entities contain other entities (such as *VariantSetMetadata*), and which contain IDs that can be used to get information from other entities (such as *Variant*'s `variantSetId`). The arrow points *from* the entity that contains the ID *to* the entity that can be identified by that ID.

FIXME: remove the Sample object from the graphic; that object isn't (yet) defined in the API.



References

References are standard genome sequences, used to provide a coordinate system for reads and variants.

References API

See References schema for a detailed reference.

References Data Model

A genome assembly is a digital representation of a genome. It is typically composed of *contigs*, each an uninterrupted string representing a DNA sequence, arranged into *scaffolds*, each of which orders and orients a set of contigs. Scaffolds are typically represented as a string, with runs of wildcard characters (N or n) used to represent interstitial regions of uncertain DNA between contigs.

A reference genome is a genome assembly that other genomes are compared to and described with respect to. For example, sequencing reads are mapped to and described with respect to a reference genome in the API, and genetic variations are described as edits to reference scaffolds/contigs. In the API a reference genome is described by a *ReferenceSet*. In turn a *ReferenceSet* is composed of a set of *Reference* objects, each which represents a scaffold or contig in the assembly. *Reference* sequences are expected to have unique names within a *ReferenceSet*. .. _ga4gh: http://genomicsandhealth.org/ .. _ga4gh_dwg: http://ga4gh.org/ .. _ga4gh_api: http://ga4gh.org/#/api

Sequence Annotations

Sequence annotations describe genomic features such as genes and exons, using terms from an established sequence ontology.

Sequence Annotations API

For the Sequence Annotation schema definitions, see Sequence Annotation schema

The Sequence Annotation Schema consists of 'Features' for discontinuous data and 'Continuous' for continuous data.

Feature Based Hierarchy

A Feature describes an interval of interest on some reference(s). It has a span from a start position to a stop position as well as descriptive data. A Feature can have a parent Feature, and can have an ordered array of child Features, which enables the construction of more complex representations in a hierarchical way.

For example, a single gene Feature may be parent to several different transcript Features. The specific exons for each transcript would have that transcript Feature as parent. The same physical exon may occur as part of two different transcript Features, but in our notation, it would be encoded as two separate exon Features, each with a different parent, both occupying the same genomic coordinates. This structure can also extend to annotating CDS, binding sites or any other sub-gene level features.

The Feature Sequence Annotation Schema

This model is similar to that used by the standard GFF3 file format.

The main differences concern the deprecation and replacement of discontinuous features, the replacing of multi-parent features with multiple copies of that feature, and the ability to impose an explicit order on child features.

In the first case, a CDS composed of multiple regions is sometimes encoded as multiple rows of a GFF3 file, each with the same feature ID. This is translated in our hierarchy into a single CDS Feature with an ordered set of CDS_region Feature children, each corresponding to a single row of the original record.

In the second case, as explained above, features with multiple parents in a GFF3 record are simply replicated and assigned a new identifier as many times as needed to ensure a unique parent for every feature.

In the final case, an explicit mechanism is provided for ordering child Features. Most of the time this ordering is trivially derived from the genomic coordinate ordering of the children, but in some biologically important cases this order can differ, such as in non-canonical splicing of exomes into transcripts (also known as back splicing - see below).

A FeatureSet is simply a collection of features from the same source. An implementer may, for example, choose to gather all Features from the same GFF3 file into a common FeatureSet.

The Continuous Sequence Annotation Schema

'Continuous' defines a format for exchanging continuous valued signal data, such as those produced experimentally (e.g. ChIP-Seq data) or through calculations (e.g. conservation scores). 'Continuous' represents numerical data in which a real value (or NaN) is associated with each base position. This data is often stored in BigWig, Wiggle or BedGraph formats.

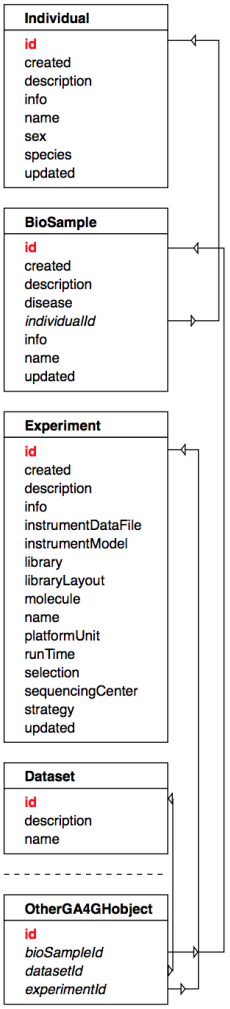
Each Continuous message consists of a start position, defined on a reference, and a list of real values. The first list value applies to the start base position and, the second list value to the base position after the start base, and so forth. The list of values can include NaN values to represent unsampled/unknown base positions. Alternatively, a set of Continuous messages (ContinuousSet), representing non-overlapping base positions, can be used, skipping all or some of the NaN values.

Annotation Design - RNA Considerations

Read data derived from RNA samples can differ from genomic read data due to the presence of non-genomic sequences. An example would be a read that spans a splice junction. It describes a contiguous sequence of reads, but a dis-continuous genomic region due to the missing intron. Feature level read assignment is further complicated by the existence of multiple splice isoforms. A read that can be definitely assigned to a particular feature (an exon in this case) may still not be definitely assigned to a particular transcript if multiple transcript share that exon. The annotation API needs to be able to report assignment at the feature level as well as aggregate assignment at the transcript or even the whole gene level if assignment is not more specific than that.

Splicing (other post-transcriptional modifications?) can occur with degrees of complexity. A 'typical' splice will result in a mature transcript with exon in positional (numerical) order in a head-to-tail orientation. Back splicing (tail-to-head) can result in transcripts with the exon order reversed (1-3-2-4 instead of 1-2-3-4) and even circular RNA. The exon order in a transcript as well as the orientation of the splice should be discoverable via the API. In a more general case, the API should allow child features to have an ordered relationship.

Metadata



Metadata allows organizing all the primary data types.

Metadata API

Goals and Scope

- standardized use of common attributes/values
- a schema of how objects relate to each other

The metadata API provides information on the primary data objects available via the GA4GH API, and facilities to organize primary data objects.

The current metadata API is immature and will evolve in future.

Metadata Records

Dataset

All GA4GH data objects are part of a *dataset*. A dataset is a data-provider-specified collection of related data of multiple types. Logically, it's akin to a folder, where it's up to the provider what goes into the folder. Individual data objects are linked by *datasetId* fields to Dataset objects.

Common Attribute Names and Formats

Throughout the schema definitions, a consistent use of common attributes should be enforced. The following list should serve as guidance for schema developers.

Attribute	Note
id	the objects ID, used for references at the level of the databas/server instance; locally unique
name	a more descriptive object label/identifier
description	a string describing aspects of the object; *not* to be used for a list or nested object
created	the time the record was created, in ISO8601 (see *Date and Time*)
updated	the time the record was updated, in ISO8601 (see *Date and Time*)

Date and Time Format Specifications

Date and time formats are specified as ISO8601 compatible strings, both for time points as well as for intervals and durations. An optional required granularity may be specified as part of the respective attributes' documentations.

Time points

The specification of a time point is given through the concatenation of

- a date in YYYY-MM-DD
- the designator "T" indicating a following time description
- the time of day in HH:MM:SS.SSS form, where "SSS" represents a decimal fraction of a second
- a time zone offset in relation to UTC

Units of time are:

- *Y* = year
- *M* = month
- *D* = day
- *H* = hour
- *M* = minute
- *S* = second
- *.S* = decimal fraction of a second

Examples

- **year (YYYY)**
  - *2015*
  - Time points with *year* granularity are both common for obfuscated personal data as well as technical metadata (e.g. year of publication of an analysis).

- **date (e.g. date of birth) in YYYY-MM-DD**
  - *2015-02-10*
  - This represents the standard way of representing a specific day, e.g. a date of birth.
- **time stamp in milliseconds in YYYY-MM-DDTHH:MM:SS.SSS**
  - *2015-02-10T00:03:42.123Z*
  - Timepoints with millisecond granularity are typical use cases for timing computer generated entries, e.g. the time of a record's update ("updateTime").

Implementations

- updated (ubiquitous object time stamp)
- created (ubiquitous object time stamp)

Durations

Durations are the most common form of time intervals. They do not refer to (e.g. start or end) time points. They are indicated with a leading "P", followed by unit delimited quantifiers. A leading "T" is required before the start of the time components. Durations do not have to be normalized; "PT50H" is equally valid as "P2T2H". A frequent use of durations in biomedical data resources are *age* values, e.g. "age at diagnosis"; but also "progression free survival", "followup" or "time to recurrence" (these are descriptive labels, which do not necessarily represent GA4GH schema use).

Examples

- **age in years in PnY**
  - *P44Y*
  - This would be the standard annotation for the commonly used age in years, without relation to a date of birth.
- **age in years and months in PnYnM**
  - *P43Y08M*
  - This represents an age with added months specification.
- **short time interval (e.g. 30min in experimental time series) in PTnM**
  - *PT30M*
  - A common use for durations is the recording of time points in time series, e.g. experimental interventions and observations (collections of cells from an in vitro treatment experiment; recurring drug doses in a chemotherapy treatment).

Time intervals

Time intervals consist of a combination of two time designators. These can be either two time points for start and end, or one time point and a leading (time point indicates end) or trailing (time point indicates start) duration. The time elements are separated by a forward slash "/".

While such anchored time intervals represent an option to capture different time features in a single value and to avoid disconnected references, in the context of the data schema, *anchored intervals* will presumably be used less frequently, with a qualitative anchor ("date of diagnosis", "time of sampling") representing the point of reference.

Examples

- **age with date of birth in YYYY-MM-DD/PnYnMnD**
  - 1967-11-21/P40Y10M05D
  - This value captures both the date of birth (here November 21, 1967)and the age (here 40ys, 10 months and 5 days) at a given time point, e.g. at the date of a medical diagnosis.
- **anchored 3 month interval, e.g. a therapy cycle in YYYY-MM-DD/YYYY-MM-DD**
  - 2015-04-18/2015-07-17
  - This example demonstrates use of an calendar anchored interval, with given start and end date. A typical example would be the use in medical records, e.g. for a treatment cycle; however, use for data exchange and mining purposes would be less common and usually served with a "duration" (see above).
- **experimental intervention of 30min in YYYY-MM-DDTHH:MM/PTnM**
  - 2014-12-31T23H45M/PT30M
  - Here is an example for a short term intervention of a 30 minutes duration, e.g. the celebratory exposure to a diluted sample of EtOH with various organic trace compounds, to celebrate the arrival of the new year.

Dataset

All GA4GH data objects are part of a *dataset*. A dataset is a data-provider-specified collection of related data of multiple types. Logically, it's akin to a folder, where it's up to the provider what goes into the folder. Individual data objects are linked by *datasetId* fields to Dataset objects.

Since the grouping of content in a dataset is determined by the data provider, users should not make semantic assumptions about that data. Subsets of the data in a dataset can be selected for analysis using other metadata or attributes.

Dataset Use Cases

For server implementors, datasets are a useful level of granularity for implementing administrative features such as access control (e.g. Data set X is public; data set Y is only available to lab Z's collaborators) and billing (e.g. the costs of hosting Dataset Y should be charged to lab Z).

For data curators, datasets are 'the simplest thing that could possibly work' for grouping data (e.g. Dataset X has all the reads, variants, and expression levels for a particular research project; Dataset Y has all the work product from a particular grant).

For data accessors, datasets are a simple way to scope exploration and analysis (e.g. "Are there any supporting examples in 1000genomes?"; "What is the distribution of that result in the data from our project?").

BioMetadata: *Biosample* Object

Biosample in the GA4GH Schema

The majority of use cases of GA4GH schema compatible resources will serve to facilitate the retrieval of *molecular features* (DNA sequence variations, gene expression, protein variants) measured by performing *experimental* (whole genome sequencing, expression arrays, mass spectroscopy) in conjunction with *bioinformatics* procedures, applied to a preparation of target molecules (e.g. DNA, RNA) which has been extracted from a *biological sample* (e.g. tissue biopsy, single cell from FACS, environmental sample).

In the GA4GH schema, a *Biosample* represents the main "biological item" against which molecular variants are referenced.

Biosample attributes

Attribute	Notes
id	• the Biosample's id • unique in the context of the server • used for referencing this Biosample
name	• a human readable object label/identifier • not to be used for referencing
description	• additional, unstructured information about this Biosample
disease	• OntologyTerm annotating the disease of the sample
individualId	• the *id* of the *Individual* this Biosample was derived from
created	• the time the record was created, in ISO8601
updated	• the time the record was updated, in ISO8601
attributes	• additional, structured information

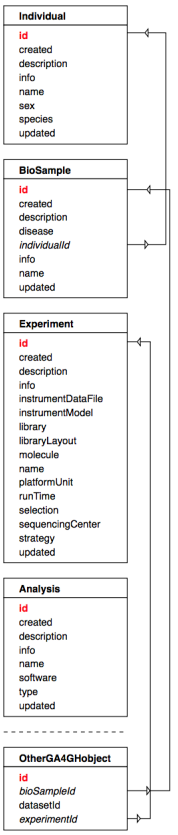
BioMetadata: *Individual* Object

Individual in the GA4GH Schema

An *Individual* is a GA4GH data object representing a biological instance (most commonly a human being or other individual organism) on whose *Biosamples* experimental analyses are performed.

Individual attributes

Attribute	Notes
id	• the Individual's id • unique in the context of the server • used for referencing this Individual
name	• a human readable object label/identifier • not to be used for referencing
description	• additional, unstructured information about this Individual
species	• OntologyTerm representing the species (NCBITaxon:9606)
sex	• OntologyTerm for the genetic sex of this individual.
created	• the time the record was created, in ISO8601
updated	• the time the record was updated, in ISO8601
attributes	• additional, structured information



AssayMetadata: *Experiment* Object

Experiment in the GA4GH Schema

Experiment attributes

Attribute	Notes
id	• the Experiment's id
•	• unique in the context of the server
•	• used for referencing this Experiment
name	• a human readable object label/identifier
•	• not to be used for referencing
description	• additional, unstructured information about this Experiment
created	• the time the record was created, in ISO8601
updated	• the time the record was updated, in ISO8601
attributes	• additional, structured information

AssayMetadata: *Analysis* Object

Analysis in the GA4GH Schema

Analysis attributes

Attribute Notes ========== ========== *id* * the Analysis's id - * unique in the context of the server - * used for referencing this Analysis *name* * a human readable object label/identifier - * not to be used for referencing *description* * additional, unstructured information about this Analysis *created* * the time the record was created, in ISO8601 *updated* * the time the record was updated, in ISO8601 *attributes* * additional, structured information ========== ==========

Allele Annotations

Allele annotations are additional pieces of data often generated by algorithms which help to describe, classify, and understand variants.

Allele Annotation API

See Allele Annotation schema for a detailed reference.

Introduction

Variant alleles can be annotated by comparing them to gene annotation data using a variety of algorithms. A standard form of annotation is to compare alleles to a transcript set and calculate the expected functional consequence of the change (e.g. a variant within a protein coding transcript may change the amino acid sequence of the resulting protein).

This API supports the mining of variant annotations by region and the filtering of the results by predicted functional effect.

Allele Annotation Schema Entities

The `VariantAnnotation` data model, is based on the results provided by variant annotation programs such as VEP, SnpEff and Annovar and others, as well as the VCF's ANN format .

Record	Description
VariantAnnotationSet	A VariantAnnotationSet record groups VariantAnnotation records. It represents the comparison of a VariantSet to specified gene annotation data using specified algorithms. It holds information describing the software and annotation data versions used.
VariantAnnotation	A VariantAnnotation record represents the result of comparing a single variant to the set of annotation data. It contains structured sub-records and a flexible key-value pair 'info' field.
TranscriptEffect	A TranscriptEffect record describes the effect of an allele on a transcript.
AlleleLocation	An AlleleLocation record holds the location of an allele relative to a non-genomic coordinate system such as a CDS or protein. It holds the reference and alternate sequence where appropriate
HGVSAnnotation	A HGVSAnnotation record holds Human Genome Variation Society (HGVS) descriptions of the sequence change at genomic, transcript and protein level where relevant.
AnalysisResult	An AnalysisResult record holds the output of a prediction package such as SIFT on a specific allele.

The schema is shown in the diagram below.

TranscriptEffect attributes

A `VariantAnnotation` record may have many `TranscriptEffect` records as one is reported for each possible combination of alternate alleles and overlapping transcripts. The record includes:

- The identifier of the transcript feature the variant was analysed against.
- The alternate allele of the variant analysed. This is necessary as the current variant model supports multiple alternate alleles.
- The predicted effects of the allele on the transcript, which should be described using Sequence Ontology terms.
- A `HGVSAnnotation` record containing variant descriptions at all relevant levels.
- `AlleleLocation` records describing the changes at cDNA, CDS and protein level.
- A set of results from prediction packages analyzing the allele impact.

Search Options

VariantAnnotationSets can be extracted by Dataset or VariantSet, or retrieved by id.

A VariantAnnotationSet can be searched for VariantAnnotations by region and filters can be applied.

- A region to search must be specified. This can be done by providing a reference sequence (identified by name or id) with start and end coordinates.
- Results can be filtered by the predicted effect of the variant using a Sequence Ontology OntologyTerm.

RNA Quantification

The RNA quantifications provides a means of obtaining feature level quantifications derived from a set of RNA reads.

RNA Quantification API

For the RNA Quantification schema definitions, see the RNA Quantification schema

RNA Quantification

The RNA Quantification provides a means of obtaining feature level quantifications derived from a set of RNA reads.

Case 1: Obtain quantification data for one or more features (genes) in an RNASeq experiment

User desires: Feature quantification data (numeric) for one or more features identified in an RNASeq experiment result. User will provide a list of one or more features for which results should be returned. If a feature list is not provided, all quantification results for the selected RNASeq experiment should be returned. Numeric quantification should be provided as raw read count to allow for user conversion to desired units. If desired TPM, RPKM or other enumerated units of measure can also be reported.

Additional Considerations: An RNASeq experiment result is the output of running an analysis pipeline on a set of read data. The result should have metadata that describes the pipeline that was used in detail. It should include the identity of the input reads in enough detail to retrieve the read data. All software used should include version, parameters and command line. Any genome or transcriptome annotations used should be described in enough detail to retrieve the exact version used. Any changes in software, parameters, annotations or other analysis details should result in a new RNASeq experiment result associated with that input read data.

Case 2: Obtain quantification data for one or more features (genes) for comparison between multiple RNASeq experiments

User desires: Feature quantification data (numeric) for one or more features identified in one or more RNASeq experiment results. User will provide a list of one or more features for which results should be returned. If a feature list is not provided, all quantification results for the selected RNASeq experiment(s) should be returned. User will provide a list of one or more RNASeq experiments to obtain quantification results from.

Additional Considerations: In order to request quantifications for comparison with either repository datasets or a local dataset the user needs to determine if the repository RNASeq experiment is comparable to other datasets. Pipeline metadata needs to be provided so that this can be determined: Sample-level: bio sample data such as tissue type, collection methods, sample preparation protocol, library generation protocol, spikes used Read-level: sequencer type and protocol, sequence data generation software version, parameters and command line Quantification-level: genome annotation, transcriptome annotation if any, software pipeline including versions, parameters and command line Batch-level: Adjustments or normalization done at the batch level

Case 3: Obtain input data to use in Assembly activities

User desires: Sequence level read data for both mapped and unmapped reads in the associated RNA experiment. For typical read data this is contained in the FASTQ file(s) produced by the sequencer pipeline. The API should either provide the original FASTQ or the read data necessary and sufficient to generate it. It is desirable to be able to easily retrieve all the related reads at the fragment level for downstream analysis. At this time, these would be either single or paired reads but for future-proofing the API should be able to handle the delivery of an arbitrary number of reads for a specific fragment.

Case 4: Obtain input data for DESeq Differential Expression analysis

User desires: Feature quantification array for two or more comparable RNASeq experimental results. This is similar to the case where the user requests feature level data. In this case, it is critical that the user be able to identify comparable datasets.

Case 5: Obtain input data for RNASeq analysis by Kallisto software

User desires: Calculate feature quantification by a new method. In the Kallisto example here, the software does not utilize read alignments. Repository needs to be able to supply raw read sequence (FASTQ format or convertible to FASTQ) and optionally annotation for the user.

Case 6: Obtain quantification data for non-read-based RNA experiments (MicroArrays)

User desires: Discover and retrieve feature level quantification data that is derived from non-read-based sources such as the large sources of microarray-based expression data. The quantification API needs to be source agnostic and allow for a general linking of quantity to feature. It must be flexible and not lock the results to a reads/sequencer data collection model. There should be no required data source or metadata fields that are specific for a given data collection method.

Annotation Design - RNA Considerations

Read data derived from RNA samples can differ from genomic read data due to the presence of non-genomic sequences. An example would be a read that spans a splice junction. It describes a contiguous sequence of reads, but a dis-continuous genomic region due to the missing intron. Feature level read assignment is further complicated by the existence of multiple splice isoforms. A read that can be definitely assigned to a particular feature (an exon in this case) may still not be definitely assigned to a particular transcript if multiple transcript share that exon. The annotation API needs to be able to report assignment at the feature level as well as aggregate assignment at the transcript or even the whole gene level if assignment is not more specific than that.

Splicing (other post-transcriptional modifications?) can occur with degrees of complexity. A 'typical' splice will result in a mature transcript with exon in positional (numerical) order in a head-to-tail orientation. Back splicing (tail-to-head) can result in transcripts with the exon order reversed (1-3-2-4 instead of 1-2-3-4) and even circular RNA. The exon order in a transcript as well as the orientation of the splice should be discoverable via the API. In a more general case, the API should allow child features to have an ordered relationship.

The annotation API needs to also be flexible enough to handle multiple references in the same gene or transcript. This is needed to cover the cases of fusion genes or inter-chromosomal translocations.

RNA Quantification Schema

The RNA Quantification Schema is designed around a quantification analysis. Each set of feature quantifications describes the results of running an analysis on a set of input data.

The RnaQuantificationSet collects a group of related RnaQuantifications. These are most likely associated by being part of a multi-sample experiment. For example, a time course experiment would be described by a RnaQuantificationSet with the individual RNASeq experiments of the time point being represented as the member RnaQuantifications.

The RnaQuantification describes the analysis pipeline used as well as the input reads dataset and which sequence annotations, if any, used.

ExpressionLevel contains the identity of the specific feature measured as well as the final resulting quantification from the pipeline.

Genotype to Phenotype

Genotype and phenotype data can be linked via evidence. This protocol provides methods for describing phenotypes and associating them with genomic features.

Genotype To Phenotype API

Summary

This API allows users to search for genotype-phenotype associations in a GA4GH datastore. The user can search for associations by building queries composed of features, phenotypes, and/or evidence terms. The API is designed to accommodate search terms specified as either a string, external identifier, ontology identifier, or as an 'entity' (See Data Model section). These terms are combined as an `AND` of `(feature && phenotype && evidence)`. This flexibility in the schema allows a variety of data to be stored in the database and allows users to express a wide range of queries.

Users will receive an array of associations as a response. Associations contain description and environment fields in addition to the relevant feature, phenotype, and evidence fields for that instance of association.

Multiple server collation - Background

G2P servers are planned to be implemented in three different contexts:

- As a wrapper around standalone local G2P "knowledge bases" (eg Monarch, CiVIC,etc). Important considerations are the API needs to function independently of other parts of the API and separately from any specific omics dataset. Often, these databases are not curated with complete Feature fields (referenceName, start, end, strand)

- Coupled with sequence annotation and GA4GH datasets. Clients will want implementation specific featureId/genotypeId to match and integrate with the rest of the APIs.

- Operating in concert with other instances of g2p servers where the client's loosely federated query is supported by heterogeneous server. Challenges: Normalizing API behavior across implementations (featureId for given region different per implementation)

Approach

We based our original work on the model captured in ga4gh/ga4gh-schemas commit of Jul 30, 2015. This version of the schema predates the separated genotype to phenotype files from baseline. After on review of the schemas and code, the team had feedback about separation of responsibility in the original API. The API was refactored to separate the searches for genotype, phenotype, feature and associations.

Data Model

The cancer genome database Clinical Genomics Knowledge Base published by the Monarch project was the source of Evidence.

Intent: The GA4GH Ontology schema provides structures for unambiguous references to ontological concepts and/or controlled vocabularies within Protocol Buffers. The structures provided are not intended for de novo modeling of ontologies, or representing complete ontologies within Protocol Buffers. References to e.g. classes from external ontologies or controlled vocabularies should be interpreted only in their original context i.e. the source ontology.

Due to the flexibility of the data model, users have a number of options for specifying each query term feature, phenotype, and evidence.

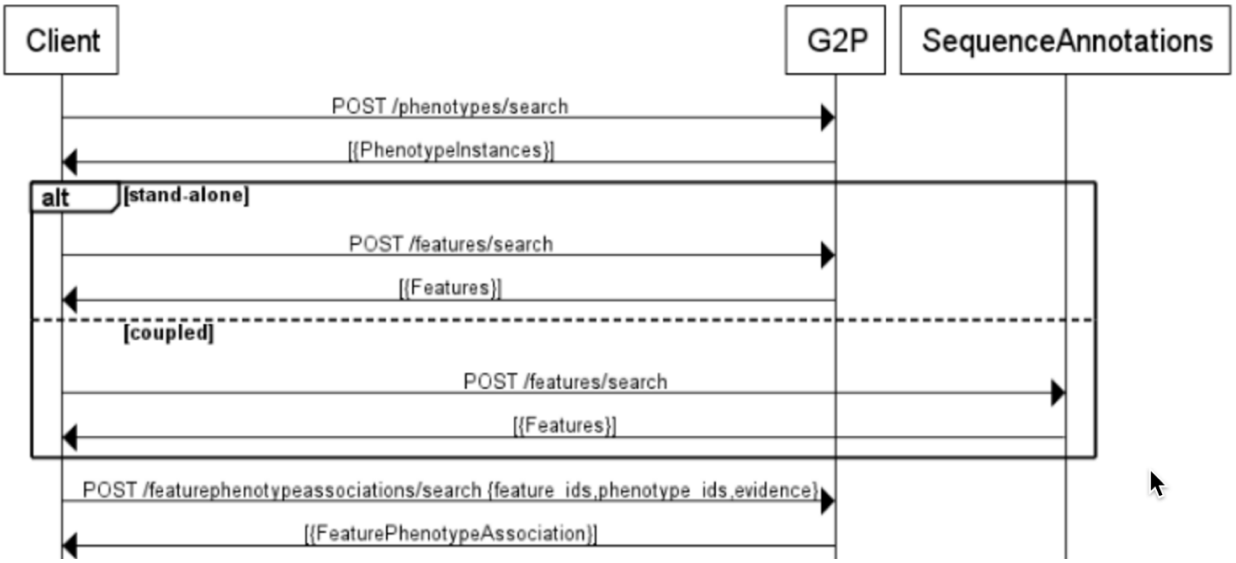
API

The G2P schemas define several endpoints broken into two entity searches and an association search.

A feature or phenotype can potentially be represented in increasing specificity as either [a string, an ontology identifier, an external identifier, or as a feature 'entity']. One criticism of the previous API is that it was overloaded, violating the design goal of separation of concerns. Specifically it combines the search for evidence with search for features & search for genotypes.

The refactored API moves search, alias matching and external identifiers lookup to dedicated end points. To separate concens, a client performs the queries for evidence in two steps: first find the desired entities and then use those enitity identifiers to narrow the search for evidence.

Additionally the API supports two implementation styles: integrated and standalone.



Entity Searches

- `/features/search`

  - Given a SearchFeaturesRequest, return matching *features* in the `current 'omics dataset`. Intended for sequence annotation and GA4GH datasets.

- `/phenotypes/search`
  - Given a SearchPhenotypesRequest, return matching *phenotypes* in the in the `current g2p dataset.`

Association Search

- `/featurephenotypeassociations/search`
  - Given a SearchGenotypePhenotypeRequest, return matching *evidence associations* in the `current g2p dataset.`

Usage

1. As a GA4GH client, use entity queries for the genotypes and phenotypes you are interested in.
2. Create an association search using the entity identifiers from step 1.
3. Repeat 1-2 as necessary, collating responses on the client.

Many types rely heavily on the concept of an OntologyTerm (see end of document for discussion on usage of OntologyTerms).

Implementation

Source Code

- Front End '/features/search', '/datasets/<datasetId>/features/search', '/phenotypes/search', '/featurephenotypeassociations/search'
- Back End 'runSearchFeatures', 'runSearchGenotypePhenotypes', 'runSearchPhenotypes', 'runSearchGenotypes'
- Datamodel 'getAssociations' Datamodel 'getAssociations' (Features)

Tests

- End to End

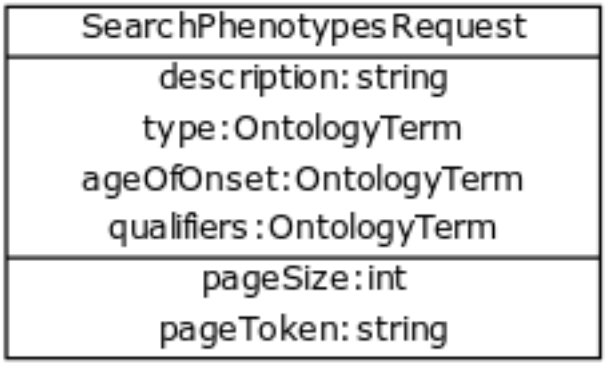
Help Wanted: Any or all use cases and scenarios

Acceptance

- Submittal of 3 simultaneous pull-requests for server, schema and compliance repositories
- 2 +1s for each repository from outside the development team
- Additional 3 day review for schemas

API Details and Examples

/phenotypes/search



Terms within a query are combined via AND e.g

```
request = "phenotype": { description:"AML",  "ageOfOnset": {"id": "http://purl.
↪obolibrary.org/obo/HP_0003581"}}

is transformed by the server to:

query = (description="AML" and ageOfOnset="http://purl.obolibrary.org/obo/HP_0003581")
```

Items in the qualifiers array are OR'd together. For example, severe or abnormal:

```
request = ... "phenotype": { description:"AML",  "qualifiers": [{"id": "http://purl.
↪obolibrary.org/obo/PATO_0000396"},{"id":"http://purl.obolibrary.org/obo/PATO_0000460
↪"}] } ....

is transformed by the server to:

query = (description="AML" and (qualifier = "http://purl.obolibrary.org/obo/PATO_
↪0000460" or qualifier = "http://purl.obolibrary.org/obo/PATO_0000460"))
```

The service returns a list of matching PhenotypeInstances.

Examples:Phenotype Lookup

Q: I have a Disease ontology id ("OBO:OMIM_606764").

Use an OntologyTerm.

```
request = { ...  "type": {"id": "http://purl.obolibrary.org/obo/OMIM_606764"}  .... }
```

The system will respond with phenotypes that match on OntologyTerm.id

Q: I have a phenotype id ("p12345") Create an PhenotypeQuery using id field.

```
request = ...  {  "id": "p12345"  } ....
```

The system will respond with phenotypes that match on PhenotypeInstance.id

Q: I have an ontology term for a phenotype (HP:0001507, 'Growth abnormality')

Use an OntologyTerm.

```
request = ...  {  "type": {"id": "http://purl.obolibrary.org/obo/HP_0001507"}  } ....
```

The system will respond with phenotypes that match on OntologyTerm.id

Q: I am only interested in phenotypes qualified with (PATO_0001899, `decreased circumference`) Create a PhenotypeQuery

```
request = ...  {  "qualifiers": [{"id": "http://purl.obolibrary.org/obo/PATO_0001899"}
↪] } ....
```

The system will respond with phenotypes whose qualifiers that match that ontology 'is_a'.

Q: I have a disease name "inflammatory bowel disease".

Create an PhenotypeQuery using description field. `{"description": "inflammatory bowel disease",...}` The system responds with Phenotypes that match on OntologyTerm.description Note that you can wildcard description. `{"description": ".*bowel.*",...}` Supported regex

---

/features/search

This endpoint is provided to serve features/variants/etc hosted by a g2p dataset when it is deployed independently of the sequenceAnnotations API. The request and response payloads are identical to `/datasets/<datasetId>/features/search`.

Terms within a query are combined via AND e.g:

```
request = { "name":"KIT",  "referenceName": "hg38" }

becomes

query = (name="KIT" and referenceName ="hg38")
```

The service returns a list of matching Features.

Examples:Genotype Lookup

Note: since we have switched to relying on the features/search API, external identifier queries have been deprecated. Refer to features/search documentation.

Q: I have a SNPid ("rs6920220"). Create an External Identifier Query.

`{... {"ids": [{"identifier": "rs6920220", "version": "*", "database": "dbSNP"}]}, ... }`

The endpoint will respond with features that match on external identifier. Multiple identifiers are OR'd together.

Q: I have an identifier for BRCA1 `GO:0070531` how do I query for feature? Create an OntologyTerm query: `{... {"type": {"id":"http://purl.obolibrary.org/obo/GO_0070531"}, ... }`

The endpoint will respond with features that match on that term.

Q: I only want somatic variant features `SO:0001777` how do I limit results? Specify featureType `{... {"featureType":"http://purl.obolibrary.org/obo/SO_0001777", ... }` The endpoint will respond with features that match on that type.

/features/search

See sequence annotations documentation.

/featurephenotypeassociations/search

The endpoint accepts a SearchGenotypePhenotypeRequest POST. The request may contain a feature, phenotype, and/or evidence, which are combined as a logical AND to query the underlying datastore. Missing types are treated as a wildcard, returning all data. The genotype and phenotype fields are either null or a list of identifiers returned from the entity queries. The evidence query object allows filtering by evidence type.

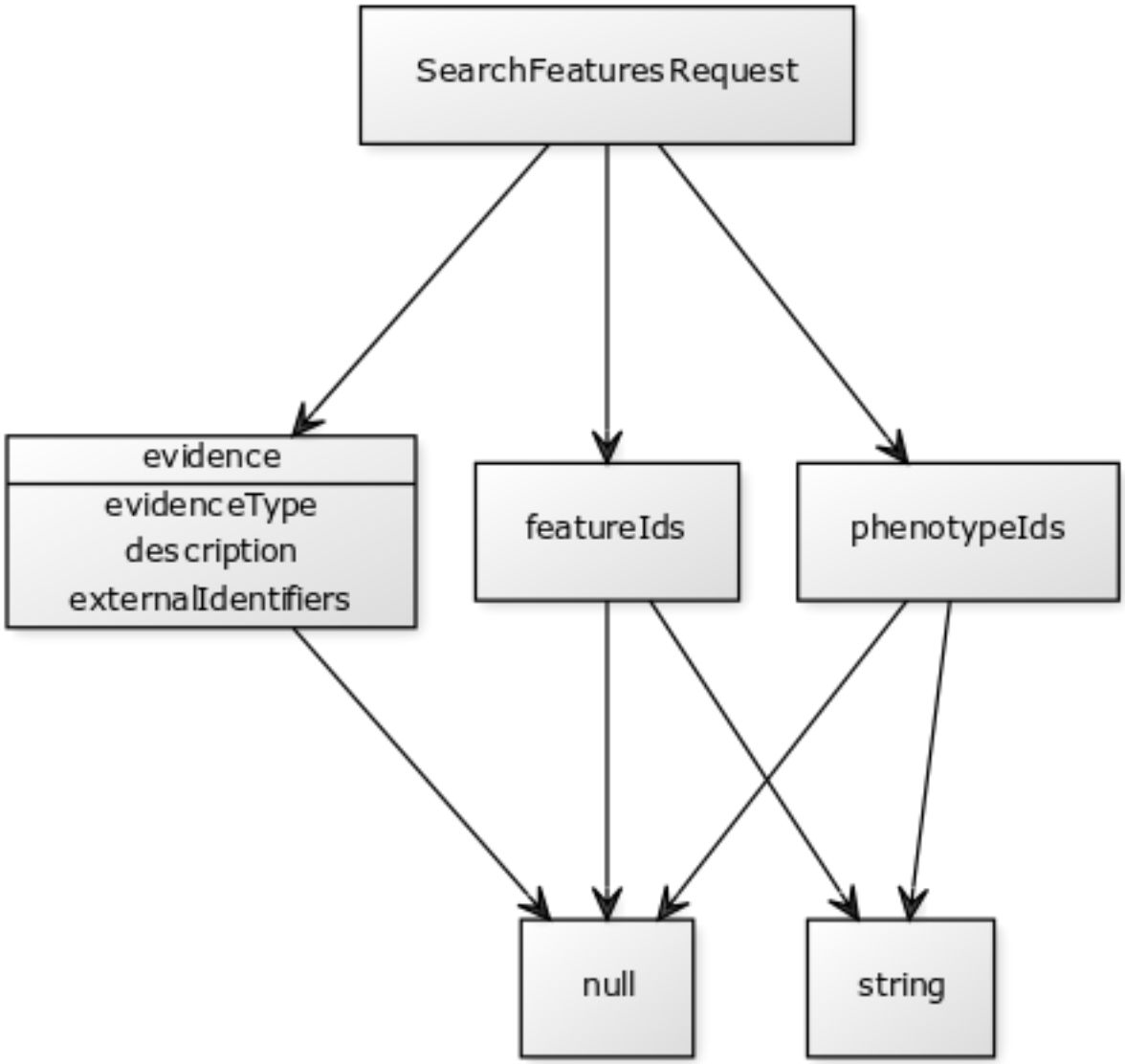


Fig. 1.1: http://yuml.me/edit/024cf70f

The SearchGenotypePhenotype search is simplified. Features and Phenotypes are expressed as a simple array of strings. Evidence can be queried via the new EvidenceQuery.

The response is returned as a list of associations.

Implementation Guidance: Results

Q: I need a place to store publication identifiers or model machine learning and statistical data.

The "info" key value pair addition to Evidence.

```
{
  "evidenceType": {
    "sourceName": "IAO",
    "id": "http://purl.obolibrary.org/obo/IAO_0000311",
    "sourceVersion": null,
    "term": "publication"
  },
  "info": {"source": ["PMID:21470995"]},
  "description": "Associated publication"
}
{
  "evidenceType": {
    "sourceName": "OBI",
    "id": "http://purl.obolibrary.org/obo/OBI_0000175",
    "sourceVersion": null,
    "term": "p-value"
  },
  "info": {"p-value": ["1.00e-21"]}
  "description": "Associated p-value"
},
{
  "evidenceType": {
    "sourceName": "OBCS",
    "id": "http://purl.obolibrary.org/obo/OBCS_0000054",
    "sourceVersion": null,
    "term": "odds ratio"
  },
  "description": "1.102"
}
```

Use cases

1. As a clinician or a genomics researcher, I may have a patient with Gastrointestinal stromal tumor, *GIST*, and a proposed drug for treatment, *imatinib*. In order to identify whether the patient would respond well to treatment with the drug, I need a list of features (e.g. genes) which are associated with the sensitivity of *GIST* to *imatinib*. Suppose I am specifically interested in a gene, *KIT*, which is implicated in the pathogenesis of several cancer types. I could submit a query to `/featurephenotypeassociations/search` with *GIST* as the phenotype, *KIT* as the feature, and `clinical study evidence <http://purl.obolibrary.org/obo/ECO_0000180>` as the evidence.

   In response, I will receive back a list of associations involving *GIST* and *KIT*, which I can filter for instances where *imatinib* is mentioned. URI's in the `associations` field could - hypothetically - be followed to discover that `GIST patients with wild-type *KIT* have decreased sensitivity to therapy with imatinib <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2651076/>`.

If I left both the `feature` and `evidence` fields as `null`, I would receive back all associations which involve GIST as a phenotype.

2. As a non-Hodgkin's lymphoma researcher, I may know that the gene *CD20* has an abnormal expression in `Hodgkin's lymphoma <http://purl.obolibrary.org/obo/DOID_8567>`. I might be interested in knowing whether *CD20* also has an abnormal expression in `non-Hodgkin lymphoma <http://purl.obolibrary.org/obo/DOID_0060060>`. Therefore I could perform a query with *CD20* as a feature, non-Hodgkin's lymphoma as a phenotype, and `RNA sequencing <http://purl.obolibrary.org/obo/OBI_0001177>` as the evidence type.
3. As a genetic counselor, I may be wondering if a mutation in one of my clients' genes has ever been associated with a disease. I could then do a query based on the gene name as the feature and `disease <http://purl.obolibrary.org/obo/DOID_4>` as the phenotype.

For specifics of the json representations, please see the `server <https://github.com/ga4gh/ga4gh-server>` and `compliance <https://github.com/ga4gh/compliance>` repositories.

Ontologies

Usage: Multiple ontology terms can be supplied e.g. to describe a series of phenotypes for a specific sample. The OntologyTerm message is not intended to model relationships between terms, or to provide mappings between ontologies for the same concept. Should an OntologyTerm be unavailable, or terms unmapped then an 'annotation' can be provided which can later be mapped to an ontology term using a service designed for this. Using OntologyTerm is preferred to using Annotation. Though annotations can be supplied with related ontology terms if desired. A use case could be when a free text annotation is very specific and a more general OntologyTerm is supplied.

Read more about Ontology Terms

Directions for future capabilities.

Flexible representation of Feature

- **Q: I need to lookup Feature by proteinName or other external id. How do look them up?** Currently, sequence annotation's features/search supports search by name or location. Future versions should implement lookup by alias/
- **Q: I have results from multiple G2P Servers. How do I collate them across datasets and implementations?** This is a subject for the investigation as we create a federation of G2P servers. The responsibility for collating features and associations across servers. One strategy might be to use HGVS' DNA annotation for as a neutral identifier for feature.

Expanding scope to entities other than Feature

Consider instead a PhenotypeAssociation which has a wider scope; the objects it connects and the evidence type determines the meaning of the association

Network

GA4GH services can communicate with each other about the services they offer over network protocols. This includes the Peer Service.

Connecting GA4GH Services

The GA4GH aims to create an easy to implement network of Genomics data services. To achieve this end, the protocol presents some message, endpoints, and methods that are dedicated to communicating about a service's status when establishing peers.

Genomic data is not passed over the Network methods. Network participation is not mandatory, although it is encouraged.

Peer Service

Two servers, if they are able to communicate over a network connection, can form a simple ad-hoc peer-to-peer (P2P) relationship by notifying each other of their existence. A network can be formed by allowing services to share their list of peers with each other.

Use Cases

- Alice would like to join an existing P2P network by announcing to a known peer. By sending an announce request with the URL of her service to the URL of the known good peer, she has advertised her services.
- Alice would like to create an ad-hoc peer-to-peer network with Bob. She first announces her service to Bob and then adds Bob's service to her list of peers.
- Alice would like to get a list of all the available datasets on a GA4GH network. She first requests the dataset from a known member of the network. She then requests that service's list of peers. She then requests the list of datasets from each peer in the peer list. By recursively following peer lists she can list all the datasets on the network.
- Alice would like to advertise the protocol version her service presents. This is done by exposing an endpoint where a client can request the protocol version and other information about her service.

Network Topology Design

Ad-hoc private networks can be established between peers, as well as hub and spoke models.

Institutions may choose to take advantage of the Peer Service to tier services. This is done by presenting a service to the public on the GA4GH network that performs aggregations of data on underlying peers. These peers only expose their services to the aggregation service.

We are seeking client demonstrations of crawlers, authentication mechanisms, and aggregators that take advantage of these methods.

Network Diagram

This diagram conceives of a network architecture where public nodes create a fully connected network, while aggregators over private data make some of these results available to the wider scientific community.

This architecture is not enforced by the protocol and network participants will determine what topology the network takes.

Network Membership

Service operators choose whether to respond to announcements, or whether to add a peer to their peer list. Since services are free to manage their peer list as they please, various network configurations can be achieved. A single decentralized fully connected network can be made by bootstrapping using a list of known good peers.

Public Initial Peers

The GA4GH attempts to bootstrap this network by maintaining the latest released network protocol at http://1kgenomes.ga4gh.org . However, the process of evaluating announcements requires human curation, so do not expect your peer to be listed immediately.

Private Networks

"White listing" allows one to create a service that only responds to requests from known hosts. By configuring a node to only respond to requests from a certain domain, it is placed in a private network.

By "white listing" only the peers on a service's peer list, it is possible to create private peer to peer networks. These nodes will only respond to requests from the managed list of peers. This is considered a detail of implementation, the protocol itself does not enforce this.

Methods

The Peer Service presents three endpoints: `/announce`, `peers/list`, and `/info`. Small messages about services or potential peers are communicated over them.

Announcements

Any client can notify a server about a possible peer using an AnnouncePeerRequest, which is a simple message including the URL of the intended peer. That service can then respond to the announcement by adding that peer to its list of peers.

By reviewing announcements a server operator can control which announcements are promoted into the list of connect peers.

Listing Peers

Each service, in addition to receiving announcements about the presence of other peers, shares its list of peers at the `peers/list` endpoint. This list can be paged through similar to other GA4GH endpoints. Each entry in the ListPeersResponse includes a URL to a possible peer. It is up to individual services to maintain their list of peers.

The endpoint is named off of the `peers` path because, in practice, we expect implementations to provide `peers/add` and `peers/remove` methods, however, these are internal configuration paths and not needed to comply with the protocol.

Get Info

To assist in the process of evaluating a peer, an info endpoint allows a client to request information about the service. The GetInfoResponse includes the protocol version.

For implementation details, please visit the protobuf description.

Installing the GA4GH Schemas

The schemas are documents (text files) that formally describe the messages that pass between GA4GH servers and clients, which we also refer to collectively as "the API." The schemas are written in a language called Protocol Buffers 3.

For instructions on how to install Protocol Buffers 3 on your system, consult the protocol buffers C++ installation instructions page.

We use the schemas in a couple of different ways:

- to generate source code
- to generate documentation

Generating Source Code

```
$ cd src/main/proto && protoc --python_out=. ga4gh/*
```

Installing the Documentation Tools and Generating Documentation

We use a tool called Sphinx to generate the documentation from Protocol Buffers input files.

Install prerequisites

To use the Sphinx/Protocol Buffers documentation generator, you must install some software packages it requires.

Maven

We use the Maven build tool to control processing the schemas. Installing Maven will also install Java.

Ubuntu

```
$ sudo apt-get install maven
```

CentOS/Fedora

```
$ sudo yum install maven
```

Mac OS X

```
$ brew install maven
```

Python/PIP

We use some Python utility programs also.

Install the Python installer `pip`.

Ubuntu

```
$ sudo apt-get install python-pip
```

CentOS/Fedora

```
$ sudo yum install python-pip
```

Mac OS X

Use `brew`:

```
$ brew install pip
```

Or download `pip` from here and run it:

```
$ python get-pip.py
```

Putting it all together

Do this once to install all required Python packages:

```
$ sudo pip install -r python/dev_requirements.txt -c python/constraints.txt
```

Generate the documentation

With those prerequisites out of the way, do this anytime you wish to generate the documentation.

Assuming your working directory is the base "`schemas`" directory, do this:

```
$ mvn site
```

The documentation you generate will reside in `target/generated-docs/merged/html`. To view it, open the file `target/generated-docs/merged/html/index.html` in a browser. An automatically generated UML diagram of the current schema can be seen below. Please open it a new tab for details. Containments are indicated by orange lines, whereas Id-references are indicated by dashed green lines. Click the image below to open it in a separate browser window. You should be able to zoom in (command+ or control+ on most browsers) and scroll around the image to see the details.

Schemas

common

This file defines common types used in other parts of the schema. There are no directly associated methods.

enum `Strand`

Symbols STRAND_UNSPECIFIED|NEG_STRAND|POS_STRAND

Indicates the associated DNA strand for some data item.

•*STRAND_UNSPECIFIED*: If no strand data is available.

•*NEG_STRAND*: The negative (-) strand.

•*POS_STRAND*: The postive (+) strand.

enum `NullValue`

Symbols NULL_VALUE

•*NULL_VALUE*: Null value.

message `GAException`

Fields

- **`error_code`** (*`integer`*) – Numerical error code
- **`message`** (*`string`*) – The error message.

When returning an HTTP error response, a server may also return a JSON formatted GAException to better describe the error.

message `Position`

Fields

- **`reference_name`** (*`string`*) – The name of the *Reference* on which the *Position* is located.
- **`position`** (*`long`*) – The 0-based offset from the start of the forward strand for that *Reference*. Genomic positions are non-negative integers less than *Reference* length.
- **`strand`** (*`Strand`*) – Strand the position is associated with.

A *Position* is an unoriented base in some *Reference*. A *Position* is represented by a *Reference* name, and a base number on that *Reference* (0-based).

message `ExternalIdentifier`

Fields

- **`database`** (*`string`*) – The source of the identifier, e.g. *Ensembl*.
- **`identifier`** (*`string`*) – The ID defined by the external database, e.g. *ENST00000000000*.
- **`version`** (*`string`*) – The version of the object or the database, e.g. *78*.

Identifier from a public database

message `Experiment`

Fields

- **id** (*string*) – The experiment ID
- **name** (*string*) – The name of the experiment.
- **description** (*string*) – A description of the experiment.
- **message_create_time** (*string*) – The time at which this message was created. Format: ISO 8601, YYYY-MM-DDTHH:MM:SS.SSS (e.g. 2015-02-10T00:03:42.123Z)
- **message_update_time** (*string*) – The time at which this message was last updated. Format: ISO 8601, YYYY-MM-DDTHH:MM:SS.SSS (e.g. 2015-02-10T00:03:42.123Z)
- **run_time** (*string*) – The time at which this experiment was performed. Granularity here is variable (e.g. date only). Format: ISO 8601, YYYY-MM-DDTHH:MM:SS (e.g. 2015-02-10T00:03:42)
- **molecule** (*string*) – The molecule examined in this experiment. (e.g. genomics DNA, total RNA)
- **strategy** (*string*) – The experiment technique or strategy applied to the sample. (e.g. whole genome sequencing, RNA-seq, RIP-seq)
- **selection** (*string*) – The method used to enrich the target. (e.g. immunoprecipitation, size fractionation, MNase digestion)
- **library** (*string*) – The name of the library used as part of this experiment.
- **library_layout** (*string*) – The configuration of sequenced reads. (e.g. Single or Paired).
- **instrument_model** (*string*) – The instrument model used as part of this experiment. This maps to sequencing technology in BAM.
- **instrument_data_file** (*string*) – The data file generated by the instrument. TODO: This isn't actually a file is it? Should this be *instrumentData* instead?
- **sequencing_center** (*string*) – The sequencing center used as part of this experiment.
- **platform_unit** (*string*) – The platform unit used as part of this experiment. This is a flowcell-barcode or slide unique identifier.
- **attributes** (*Attributes*) – A map of additional experiment information.

An experimental preparation of a sample.

message Analysis

Fields

- **id** (*string*) – Formats of id | name | description | accessions are described in the documentation on general attributes and formats.
- **name** (*string*) –
- **description** (*string*) –
- **created** (*string*) – The time at which this record was created, in ISO 8601 format.
- **updated** (*string*) – The time at which this record was last updated, in ISO 8601 format.
- **type** (*string*) – The type of analysis.
- **software** (*string*) – The software run to generate this analysis.

- **attributes** (*Attributes*) – A map of additional analysis information.

An analysis contains an interpretation of one or several experiments. (e.g. SNVs, copy number variations, methylation status) together with information about the methodology used.

message `OntologyTerm`

Fields

- **term_id** (*string*) – Ontology term identifier - the CURIE for an ontology term. It differs from the standard GA4GH schema's *id* in that it is a CURIE pointing to an information resource outside of the scope of the schema or its resource implementation.
- **term** (*string*) – Ontology term - the label of the ontology term the termId is pointing to.

An ontology term describing an attribute. (e.g. the phenotype attribute 'polydactyly' from HPO)

message `Program`

Fields

- **command_line** (*string*) – The command line used to run this program.
- **id** (*string*) – The user specified ID of the program.
- **name** (*string*) – The name of the program.
- **prev_program_id** (*string*) – The ID of the program run before this one.
- **version** (*string*) – The version of the program run.

A Program describes software used in data processing or analysis.

message `AttributeValue`

Fields

- **value** (string | long | integer | boolean | double | *ExternalIdentifier* | *OntologyTerm* | *Experiment* | *Program* | *Analysis* | *NullValue* | *Attributes* | *AttributeValueList*) –

Type defining a collection of attributes associated with various protocol records. Each attribute is a name that maps to an array of one or more values. Values are chosen from both internal protobuf types and GA4GH. Values should be split into array elements instead of using a separator syntax that needs to parsed.

message `AttributeValueList`

Fields

- **values** (list of *AttributeValue*) –

message `Attributes`

Fields

- **attr** (map< string , *AttributeValueList* >) –

bio_metadata

An individual (or subject) typically corresponds to an individual human or another organism.

A Biosample refers to a unit of biological material from which the substrate molecules (e.g. genomic DNA, RNA, proteins) for molecular analyses (e.g. sequencing, array hybridisation, mass-spectrometry) are extracted. Examples would be a tissue biopsy, a single cell from a culture for single cell genome sequencing or a protein fraction from a gradient centrifugation. Several instances (e.g. technical replicates) or types of experiments (e.g. genomic array as well as RNA-seq experiments) may refer to the same

Biosample. In the context of the GA4GH metadata schema, Biosample constitutes the central reference object.

The age object permits both the (considered default) encoding of an age value in ISO8601, with arbitrary granularity; and the representation of an "age class" as qualitative ontology term. If available, a quantitative value should be used & take precedence over the age class, and class assignment should be performed at user/API level.

message `Individual`

Fields

- **`id`** (*`string`*) – The Individual's *id*. This is unique in the context of the server instance.
- **`dataset_id`** (*`string`*) – The ID of the dataset this Individual belongs to.
- **`name`** (*`string`*) – The Individual's name. This is a label or symbolic identifier for the individual.
- **`description`** (*`string`*) – The Individual's description. This attribute contains human readable text. The "description" attributes should not contain any structured data.
- **`created`** (*`string`*) – The *ISO 8601* time at which this Individual's record was created.
- **`updated`** (*`string`*) – The *ISO 8601* time at which this Individual record was updated.
- **`species`** (*`OntologyTerm`*) – For a representation of an NCBI Taxon ID as an OntologyTerm, see NCBITaxon Ontology http://www.obofoundry.org/ontology/ncbitaxon.html For example, 'Homo sapiens' has the ID 9606. The NCBITaxon ontology ID for this is NCBITaxon:9606, and therefore: species : { term_id: "NCBITaxon:9606", term : "Homo sapiens" }
- **`sex`** (*`OntologyTerm`*) – The genetic sex of this individual. Use *null* when unknown or not applicable. Recommended: PATO http://purl.obolibrary.org/obo/PATO_0020000 ... Example: sex : { term_id: "PATO:0020000", term : "female genetic sex" }
- **`attributes`** (*`Attributes`*) – A map of additional information regarding the Individual.

message `Biosample`

Fields

- **`id`** (*`string`*) – The Biosample *id*. This is unique in the context of the server instance.
- **`dataset_id`** (*`string`*) – The ID of the dataset this Biosample belongs to.
- **`name`** (*`string`*) – The Biosample's name This is a label or symbolic identifier for the biosample.
- **`description`** (*`string`*) – The biosample's description. This attribute contains human readable text. The "description" attributes should not contain any structured data.
- **`disease`** (*`OntologyTerm`*) – OntologyTerm describing the primary disease associated with this Biosample.
- **`created`** (*`string`*) – The *ISO 8601* time at which this Biosample record was created.
- **`updated`** (*`string`*) – The *ISO 8601* time at which this Biosample record was updated.
- **`individual_id`** (*`string`*) – The individual this biosample was derived from.
- **`attributes`** (*`Attributes`*) – A map of additional information about the Biosample.
- **`individual_age_at_collection`** (*`Age`*) – An age object describing the age of the individual this biosample was derived from at the time of collection. The Age object allows

the encoding of the age either as ISO8601 duraion or time interval (preferred), or as ontology term object.

message `Age`

Fields

- **`age`** (*`string`*) – The *ISO 8601* age of this object as ISO8601 duration or time intervals. The use of time intervals makes an additional anchor unnecessary (i.e. DOB and age can be represented as start-anchored time interval, e.g. 1967-11-21/P40Y10M05D) TODO: Anonymous reference time attribute/pointer?
- **`age_class`** (*`OntologyTerm`*) – An age class, e.g. corresponding to the use of "age of onset" in HPO. HPO is recommended, for example, subclasses of http://purl.obolibrary.org/obo/HP_0003674 Example: age_class : { term_id : "HP:0003596", term : "Middle age onset" }

bio_metadata_service

`GetIndividual`(*request*)

Parameters **`request`** – *`GetIndividualRequest`*

Return type *`Individual`*

Throws *`GAException`*

Gets an *Individual* by ID. *GET /individuals/{id}* will return a JSON version of *Individual*.

`SearchBiosamples`(*request*)

Parameters **`request`** – *`SearchBiosamplesRequest`*

Return type *`SearchBiosamplesResponse`*

Throws *`GAException`*

POST /biosamples/search must accept a JSON version of *SearchBiosamplesRequest* as the post body and will return a JSON version of *SearchBiosamplesResponse*.

`GetBiosample`(*request*)

Parameters **`request`** – *`GetBiosampleRequest`*

Return type *`Biosample`*

Throws *`GAException`*

Gets a *Biosample* by ID. *GET /biosamples/{id}* will return a JSON version of *Biosample*.

`SearchIndividuals`(*request*)

Parameters **`request`** – *`SearchIndividualsRequest`*

Return type *`SearchIndividualsResponse`*

Throws *`GAException`*

Gets a list of Individuals accessible through the API. *POST /individuals/search* must accept a JSON version of *SearchIndividualsRequest* as the post body and will return a JSON version of *SearchIndividualsResponse*.

message `SearchIndividualsRequest`

Fields

- **`dataset_id`** (*`string`*) – Optionally specify the dataset to search within.

- **name** (*string*) – Returns Individuals with the given name found by case-sensitive string matching.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *nextPageToken* from the previous response.

********** /individuals ********** This request maps to the body of *POST /individuals/search* as JSON.

message GetIndividualRequest

Fields

- **individual_id** (*string*) – The ID of the individual requested

This request maps to the URL *GET /individuals/{individual_id}*.

message GetBiosampleRequest

Fields

- **biosample_id** (*string*) – The ID of the biosample requested

This request maps to the URL *GET /biosamples/{biosample_id}*.

message SearchIndividualsResponse

Fields

- **individuals** (list of *Individual*) – The list of individuals.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /individuals/search* expressed as JSON.

message SearchBiosamplesRequest

Fields

- **dataset_id** (*string*) – Optionally specify the dataset to search within.
- **name** (*string*) – Returns Biosamples with the given name found by case-sensitive string matching.
- **individual_id** (*string*) – Returns Biosamples for the provided individual ID.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *nextPageToken* from the previous response.

********** /biosamples ********** This is the request sent to *POST /biosamples/search* expressed as JSON.

message SearchBiosamplesResponse

Fields

- **biosamples** (list of *Biosample*) – The list of biosamples.

- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /biosamples/search* expressed as JSON.

metadata

message Dataset

Fields

- **id** (*string*) – The dataset's id, locally unique to the server instance.
- **name** (*string*) – The name of the dataset.
- **description** (*string*) – Additional, human-readable information on the dataset.
- **attributes** (*Attributes*) – A map of additional dataset information.

A Dataset is a collection of related data of multiple types. Data providers decide how to group data into datasets. See [Metadata API](../api/metadata.html) for a more detailed discussion.

metadata_service

********** /datasets **********

SearchDatasets(*request*)

Parameters **request** – *SearchDatasetsRequest*

Return type *SearchDatasetsResponse*

Throws *GAException*

Gets a list of *Dataset* matching the search criteria.

POST /datasets/search must accept a JSON version of *SearchDatasetsRequest* as the post body and will return a JSON version of *SearchDatasetsResponse*.

GetDataset(*request*)

Parameters **request** – *GetDatasetRequest*

Return type *Dataset*

Throws *GAException*

Gets a *Dataset* by ID.

GET /datasets/{dataset_id} will return a JSON version of *Dataset*.

message SearchDatasetsRequest

Fields

- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request maps to the body of *POST /datasets/search* as JSON.

message `SearchDatasetsResponse`

Fields

- **`datasets`** (list of *`Dataset`*) – The list of datasets.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /datasets/search* expressed as JSON.

message `GetDatasetRequest`

Fields

- **`dataset_id`** (*`string`*) – The ID of the *Dataset* to be retrieved.

This request maps to the URL *GET /datasets/{dataset_id}*.

reads

This file defines the objects used to represent a reads and alignments, most importantly ReadGroupSet, ReadGroup, and ReadAlignment. See {TODO: LINK TO READS OVERVIEW} for more information.

NB: we require that all readgroups in the set are mapped to the same referenceSet.

message `ReadStats`

Fields

- **`aligned_read_count`** (*`long`*) – The number of aligned reads.
- **`unaligned_read_count`** (*`long`*) – The number of unaligned reads.
- **`base_count`** (*`long`*) – The total number of bases. This is equivalent to the sum of *alignedSequence.length* for all reads.

ReadStats can be used to provide summary statistics about read data.

message `ReadGroup`

Fields

- **`id`** (*`string`*) – The read group ID.
- **`dataset_id`** (*`string`*) – The ID of the dataset this read group belongs to.
- **`name`** (*`string`*) – The read group name.
- **`description`** (*`string`*) – The read group description.
- **`sample_name`** (*`string`*) – A name for the sample this read group's data were generated from. This field contains an arbitrary string, typically corresponding to the SM tag in a BAM file.
- **`biosample_id`** (*`string`*) – The Biosample this read group's data was generated from.
- **`experiment`** (*`Experiment`*) – The experiment used to generate this read group.
- **`predicted_insert_size`** (*`integer`*) – The predicted insert size of this read group.
- **`created`** (*`long`*) – The time at which this read group was created in milliseconds from the epoch.
- **`updated`** (*`long`*) – The time at which this read group was last updated in milliseconds from the epoch.

- **stats** (*ReadStats*) – Statistical data on reads in this read group.
- **programs** (list of *Program*) – Program can be used to track the provenance of how read data was generated.
- **reference_set_id** (*string*) – The ID of the reference set to which the reads in this read group are aligned. Required if there are any read alignments.
- **attributes** (*Attributes*) – A map of additional information about the Read Group.

A ReadGroup is a set of reads derived from one physical sequencing process.

message ReadGroupSet

Fields

- **id** (*string*) – The read group set ID.
- **dataset_id** (*string*) – The ID of the dataset this read group set belongs to.
- **name** (*string*) – The read group set name.
- **stats** (*ReadStats*) – Statistical data on reads in this read group set.
- **read_groups** (list of *ReadGroup*) – The read groups in this set.
- **attributes** (*Attributes*) – A map of additional information about the Read Group Set.

A ReadGroupSet is a logical collection of ReadGroups. Typically one ReadGroupSet represents all the reads from one experimental sample.

message LinearAlignment

Fields

- **position** (*Position*) – The position of this alignment.
- **mapping_quality** (*integer*) – The mapping quality of this alignment, meaning the likelihood that the read maps to this position.

  Specifically, this is -10 log10 Pr(mapping position is wrong), rounded to the nearest integer.
- **cigar** (list of *CigarUnit*) – Represents the local alignment of this sequence (alignment matches, indels, etc) versus the reference.

A linear alignment describes the alignment of a read to a Reference, using a position and CIGAR array.

message ReadAlignment

Fields

- **id** (*string*) – The read alignment ID. This ID is unique within the read group this alignment belongs to.

  For performance reasons, this field may be omitted by a backend. If provided, its intended use is to make caching and UI display easier for genome browsers and other lightweight clients.
- **read_group_id** (*string*) – The ID of the read group this read belongs to. (Every read must belong to exactly one read group.)
- **fragment_name** (*string*) – The fragment name. Equivalent to QNAME (query template name) in SAM.
- **improper_placement** (*boolean*) – The orientation and the distance between reads from the fragment are inconsistent with the sequencing protocol (inverse of SAM flag 0x2).

- **duplicate_fragment** (*boolean*) – The fragment is a PCR or optical duplicate (SAM flag 0x400).
- **number_reads** (*integer*) – The number of reads in the fragment (extension to SAM flag 0x1).
- **fragment_length** (*integer*) – The observed length of the fragment, equivalent to TLEN in SAM.
- **read_number** (*integer*) – The read ordinal in the fragment, 0-based and less than numberReads. This field replaces SAM flag 0x40 and 0x80 and is intended to more cleanly represent multiple reads per fragment.
- **failed_vendor_quality_checks** (*boolean*) – The read fails platform or vendor quality checks (SAM flag 0x200).
- **alignment** (*LinearAlignment*) – The alignment for this alignment message. This field will be null if the read is unmapped.
- **secondary_alignment** (*boolean*) – Whether this alignment is secondary. Equivalent to SAM flag 0x100. A secondary alignment represents an alternative to the primary alignment for this read. Aligners may return secondary alignments if a read can map ambiguously to multiple coordinates in the genome.

  By convention, each read has one and only one alignment where both secondaryAlignment and supplementaryAlignment are false.
- **supplementary_alignment** (*boolean*) – Whether this alignment is supplementary. Equivalent to SAM flag 0x800. Supplementary alignments are used in the representation of a chimeric alignment. In a chimeric alignment, a read is split into multiple linear alignments that map to different reference contigs. The first linear alignment in the read will be designated as the representative alignment; the remaining linear alignments will be designated as supplementary alignments. These alignments may have different mapping quality scores.

  In each linear alignment in a chimeric alignment, the read will be hard clipped. The *alignedSequence* and *alignedQuality* fields in the alignment message will only represent the bases for its respective linear alignment.
- **aligned_sequence** (*string*) – The bases of the read sequence contained in this alignment record (equivalent to SEQ in SAM).

  alignedSequence and *alignedQuality* may be shorter than the full read sequence and quality. This will occur if the alignment is part of a chimeric alignment, or if the read was trimmed. When this occurs, the CIGAR for this read will begin/end with a hard clip operator that will indicate the length of the excised sequence.
- **aligned_quality** (*integer*) – The quality of the read sequence contained in this alignment message (equivalent to QUAL in SAM).

  alignedSequence and *alignedQuality* may be shorter than the full read sequence and quality. This will occur if the alignment is part of a chimeric alignment, or if the read was trimmed. When this occurs, the CIGAR for this read will begin/end with a hard clip operator that will indicate the length of the excised sequence.
- **next_mate_position** (*Position*) – The mapping of the primary alignment of the *(readNumber+1)%numberReads* read in the fragment. It replaces mate position and mate strand in SAM.
- **attributes** (*Attributes*) – A map of additional information about the Alignment.

Each read alignment describes an alignment with additional information about the fragment and the read. A read alignment object is equivalent to a line in a SAM file.

message `CigarUnit`

Fields

- **`operation`** (*`Operation`*) –
- **`operation_length`** (*`long`*) – The number of genomic bases that the operation runs for. Required.
- **`reference_sequence`** (*`string`*) – *referenceSequence* is only used at mismatches (*SEQUENCE_MISMATCH*) and deletions (*DELETE*). Filling this field replaces SAM's MD tag. If the relevant information is not available, this field is unset.

A single CIGAR operation.

enum `Operation`

Symbols OPERATION_UNSPECIFIED|ALIGNMENT_MATCH|INSERT|DELETE|SKIP|CLIP_SOFT|CLIP_HARD|PAD

•*OPERATION_UNSPECIFIED*:

•*ALIGNMENT_MATCH*:

•*INSERT*:

•*DELETE*:

•*SKIP*:

•*CLIP_SOFT*:

•*CLIP_HARD*:

•*PAD*:

•*SEQUENCE_MATCH*:

•*SEQUENCE_MISMATCH*:

read_service

********** /readgroupsets **********/

`SearchReads`(*request*)

Parameters **`request`** – *`SearchReadsRequest`*

Return type *`SearchReadsResponse`*

Throws *`GAException`*

Gets a list of *ReadAlignment* s for one or more *ReadGroup* s.

searchReads operates over a genomic coordinate space of reference sequence and position defined by the *Reference* s to which the requested *ReadGroup* s are aligned.

If a target positional range is specified, search returns all reads whose alignment to the reference genome *overlap* the range. A query which specifies only read group IDs yields all reads in those read groups, including unmapped reads.

All reads returned (including reads on subsequent pages) are ordered by genomic coordinate (by reference sequence, then position). Reads with equivalent genomic coordinates are returned in an unspecified order. This order must be consistent for a given repository, such that two queries for the same content (regardless of page size) yield reads in the same order across their respective streams of paginated responses.

POST /reads/search must accept a JSON version of *SearchReadsRequest* as the post body and will return a JSON version of *SearchReadsResponse*.

SearchReadGroupSets(*request*)

Parameters **request** – *SearchReadGroupSetsRequest*

Return type *SearchReadGroupSetsResponse*

Throws *GAException*

Gets a list of *ReadGroupSet* matching the search criteria.

POST /readgroupsets/search must accept a JSON version of *SearchReadGroupSetsRequest* as the post body and will return a JSON version of *SearchReadGroupSetsResponse* . Only readgroups that match an optionally supplied biosampleId will be included in the response.

GetReadGroupSet(*request*)

Parameters **request** – *GetReadGroupSetRequest*

Return type *ReadGroupSet*

Throws *GAException*

Gets a *ReadGroupSet* by ID.

GET /readgroupsets/{read_group_set_id} will return a JSON version of *ReadGroupSet*.

message SearchReadGroupSetsRequest

Fields

- **dataset_id** (*string*) – The dataset to search.
- **name** (*string*) – Only return read group sets with this name (case-sensitive, exact match).
- **biosample_id** (*string*) – Specifying the id of a Biosample record will return only readgroups with the given biosampleId.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request maps to the body of *POST /readgroupsets/search* as JSON.

TODO: Factor this out to a common API patterns section. - If searching by a resource ID, and that resource is not found, the method will return a *404* HTTP status code (*NOT_FOUND*). - If searching by other attributes, e.g. *name*, and no matches are found, the method will return a *200* HTTP status code (*OK*) with an empty result list.

message SearchReadGroupSetsResponse

Fields

- **read_group_sets** (list of *ReadGroupSet*) – The list of matching read group sets.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /readgroupsets/search* expressed as JSON.

message GetReadGroupSetRequest

Fields

- **read_group_set_id** (*string*) – The ID of the *ReadGroupSet* to be retrieved.

This request maps to the URL *GET /readgroupsets/{read_group_set_id}*.

message `SearchReadsRequest`

Fields

- **`read_group_ids`** (*`string`*) – The ReadGroups to search. At least one id must be specified.
- **`reference_id`** (*`string`*) – The reference to query. Leaving blank returns results from all references, including unmapped reads - this could be very large.
- **`start`** (*`long`*) – The start position (0-based) of this query. If a reference is specified, this defaults to 0. Genomic positions are non-negative integers less than reference length. Requests spanning the join of circular genomes are represented as two requests one on each side of the join (position 0).
- **`end`** (*`long`*) – The end position (0-based, exclusive) of this query. If a reference is specified, this defaults to the reference's length.
- **`page_size`** (*`integer`*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **`page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

********** /reads ********** This request maps to the body of *POST /reads/search* as JSON.

If a reference is specified, all queried *ReadGroup's must be aligned to 'ReferenceSet's containing that same 'Reference*. If no reference is specified, all *ReadGroup's must be aligned to the same 'ReferenceSet*.

message `SearchReadsResponse`

Fields

- **`alignments`** (list of *`ReadAlignment`*) – The list of matching alignment messages, sorted by position. Unmapped reads, which have no position, are returned last.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /reads/search* expressed as JSON.

references

message `Reference`

Fields

- **`id`** (*`string`*) – The reference ID. Unique within the repository.
- **`length`** (*`long`*) – The length of this reference's sequence.
- **`md5checksum`** (*`string`*) – The MD5 checksum uniquely representing this *Reference* as a lower-case hexadecimal string, calculated as the MD5 of the upper-case sequence excluding all whitespace characters (this is equivalent to SQ:M5 in SAM).
- **`name`** (*`string`*) – The unique name of this reference within the Reference Set (e.g. '22').
- **`source_uri`** (*`string`*) – The URI from which the sequence was obtained. Specifies a FASTA format file/string with one name, sequence pair. In most cases, clients should call the

getReferenceBases() method to obtain sequence bases for a *Reference* instead of attempting to retrieve this URI.

- **source_accessions** (*string*) – All known corresponding accession IDs in INSDC (GenBank/ENA/DDBJ) which must include a version number, e.g. *GCF_000001405.26*.
- **is_derived** (*boolean*) – A sequence X is said to be derived from source sequence Y, if X and Y are of the same length and the per-base sequence divergence at A/C/G/T bases is sufficiently small. Two sequences derived from the same official sequence share the same coordinates and annotations, and can be replaced with the official sequence for certain use cases.
- **source_divergence** (*float*) – The *sourceDivergence* is the fraction of non-indel bases that do not match the reference this message was derived from.
- **species** (*OntologyTerm*) – For a representation of an NCBI Taxon ID as an OntologyTerm, see NCBITaxon Ontology http://www.obofoundry.org/ontology/ncbitaxon.html For example, 'Homo sapiens' has the ID 9606. The NCBITaxon ontology ID for this is NCBITaxon:9606, which has the URI http://purl.obolibrary.org/obo/NCBITaxon_9606
- **attributes** (*Attributes*) – A map of additional information.

A *Reference* is a canonical assembled contig, intended to act as a reference coordinate space for other genomic annotations. A single *Reference* might represent the human chromosome 1, for instance.

Reference s are designed to be immutable.

message ReferenceSet

Fields

- **id** (*string*) – The reference set ID. Unique in the repository.
- **name** (*string*) – The reference set name.
- **md5checksum** (*string*) – Order-independent MD5 checksum which identifies this *ReferenceSet*.

  To compute this checksum, make a list of *Reference.md5checksum* for all *Reference* s in this set. Then sort that list, and take the MD5 hash of all the strings concatenated together. Express the hash as a lower-case hexadecimal string.
- **species** (*OntologyTerm*) – For a representation of an NCBI Taxon ID as an OntologyTerm, see NCBITaxon Ontology http://www.obofoundry.org/ontology/ncbitaxon.html For example, 'Homo sapiens' has the ID 9606. The NCBITaxon ontology ID for this is NCBITaxon:9606, which has the URI http://purl.obolibrary.org/obo/NCBITaxon_9606
- **description** (*string*) – Optional free text description of this reference set.
- **assembly_id** (*string*) – The remaining information is about the source of the sequences Public id of this reference set, such as *GRCh37*.
- **source_uri** (*string*) – Specifies a FASTA format file/string.
- **source_accessions** (*string*) – All known corresponding accession IDs in INSDC (GenBank/ENA/DDBJ) ideally with a version number, e.g. *NC_000001.11*.
- **is_derived** (*boolean*) – A reference set may be derived from a source if it contains additional sequences, or some of the sequences within it are derived (see the definition of *isDerived* in *Reference*).
- **attributes** (*Attributes*) – A map of additional information.

A *ReferenceSet* is a set of *Reference* s which typically comprise a reference assembly, such as *GRCh38*. A *ReferenceSet* defines a common coordinate space for comparing reference-aligned experimental data.

reference_service

ListReferenceBases(*request*)

Parameters **request** – *ListReferenceBasesRequest*

Return type *ListReferenceBasesResponse*

Throws *GAException*

Lists *Reference* bases by ID and optional range.

POST /listreferencebases will return a JSON version of *ListReferenceBasesResponse*.

GetReferenceSet(*request*)

Parameters **request** – *GetReferenceSetRequest*

Return type *ReferenceSet*

Throws *GAException*

Gets a *ReferenceSet* by ID.

GET /referencesets/{reference_set_id} will return a JSON version of *ReferenceSet*.

SearchReferenceSets(*request*)

Parameters **request** – *SearchReferenceSetsRequest*

Return type *SearchReferenceSetsResponse*

Throws *GAException*

Gets a list of *ReferenceSet* matching the search criteria.

POST /referencesets/search must accept a JSON version of *SearchReferenceSetsRequest* as the post body and will return a JSON version of *SearchReferenceSetsResponse*.

SearchReferences(*request*)

Parameters **request** – *SearchReferencesRequest*

Return type *SearchReferencesResponse*

Throws *GAException*

Gets a list of *Reference* matching the search criteria.

POST /references/search must accept a JSON version of *SearchReferencesRequest* as the post body and will return a JSON version of *SearchReferencesResponse*.

GetReference(*request*)

Parameters **request** – *GetReferenceRequest*

Return type *Reference*

Throws *GAException*

Gets a *Reference* by ID.

GET /references/{reference_id} will return a JSON version of *Reference*.

message SearchReferenceSetsRequest

Fields

- **md5checksum** (*string*) – If unset, return the reference sets for which the *md5checksum* matches this string (case-sensitive, exact match). See *ReferenceSet::md5checksum* for details.
- **accession** (*string*) – If unset, return the reference sets for which the *accession* matches this string (case-sensitive, exact match).
- **assembly_id** (*string*) – If unset, return the reference sets for which the *assemblyId* matches this string (case-sensitive, exact match).
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

********** /referencesets ********** This request maps to the body of *POST /referencesets/search* as JSON.

message SearchReferenceSetsResponse

Fields

- **reference_sets** (list of *ReferenceSet*) – The list of matching reference sets.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /referencesets/search* expressed as JSON.

message GetReferenceSetRequest

Fields

- **reference_set_id** (*string*) – The ID of the *ReferenceSet* to be retrieved.

This request maps to the URL *GET /referencesets/{reference_set_id}*.

message SearchReferencesRequest

Fields

- **reference_set_id** (*string*) – The *ReferenceSet* to search.
- **md5checksum** (*string*) – If specified, return the references for which the *md5checksum* matches this string (case-sensitive, exact match). See *ReferenceSet::md5checksum* for details.
- **accession** (*string*) – If specified, return the references for which the *accession* matches this string (case-sensitive, exact match).
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

********** /references ********** This request maps to the body of *POST /references/search* as JSON.

message SearchReferencesResponse

Fields

- **references** (list of *Reference*) – The list of matching references.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /references/search* expressed as JSON.

message GetReferenceRequest

Fields

- **reference_id** (*string*) – The ID of the *Reference* to be retrieved.

This request maps to the URL *GET /references/{reference_id}*.

message ListReferenceBasesRequest

Fields

- **reference_id** (*string*) – The ID of the *Reference* to be retrieved.
- **start** (*long*) – The start position (0-based) of this query. Defaults to 0. Genomic positions are non-negative integers less than reference length. Requests spanning the join of circular genomes are represented as two requests one on each side of the join (position 0).
- **end** (*long*) – The end position (0-based, exclusive) of this query. Defaults to the length of this *Reference*.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request retrieves a region of a reference genome when sent to the */listreferencebases* endpoint.

message ListReferenceBasesResponse

Fields

- **offset** (*long*) – The offset position (0-based) of the given sequence from the start of this *Reference*. This value will differ for each page in a paginated request.
- **sequence** (*string*) – A substring of the bases that make up this reference. Bases are represented as IUPAC-IUB codes; this string matches the regexp *[ACGTMRWSYKVHDBN]**.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

The response from *POST /listreferencebases* expressed as JSON.

variants

This file defines the objects used to represent variant calls, most importantly VariantSet, Variant, and Call. See {TODO: LINK TO VARIANTS OVERVIEW} for more information.

message VariantSetMetadata

Fields

- **key** (*string*) – The top-level key.
- **value** (*string*) – The value field for simple metadata.

- **id** (*string*) – User-provided ID field, not enforced by this API. Two or more pieces of structured metadata with identical id and key fields are considered equivalent. FIXME: If it's not enforced, then why can't it be null?
- **type** (*string*) – The type of data.
- **number** (*string*) – The number of values that can be included in a field described by this metadata.
- **description** (*string*) – A textual description of this metadata.
- **attributes** (*Attributes*) – A map of additional information about the metadata record.

This metadata represents VCF header information.

message `VariantSet`

Fields

- **id** (*string*) – The variant set ID.
- **name** (*string*) – The variant set name.
- **dataset_id** (*string*) – The ID of the dataset this variant set belongs to.
- **reference_set_id** (*string*) – The ID of the reference set that describes the sequences used by the variants in this set.
- **metadata** (list of *VariantSetMetadata*) – Optional metadata associated with this variant set. This array can be used to store information about the variant set, such as information found in VCF header fields, that isn't already available in first class fields such as "name".

A VariantSet is a collection of variants and variant calls intended to be analyzed together.

message `CallSet`

Fields

- **id** (*string*) – The call set ID.
- **name** (*string*) – The call set name.
- **biosample_id** (*string*) – The Biosample the call set data was generated from.
- **variant_set_ids** (*string*) – The IDs of the variant sets this call set has calls in.
- **created** (*long*) – The date this call set was created in milliseconds from the epoch.
- **updated** (*long*) – The time at which this call set was last updated in milliseconds from the epoch.
- **attributes** (*Attributes*) – A map of additional information about the Call Set.

A CallSet is a collection of calls that were generated by the same analysis of the same sample.

message `Call`

Fields

- **call_set_name** (*string*) – The name of the call set this variant call belongs to. If this field is not present, the ordering of the call sets from a *SearchCallSetsRequest* over this *VariantSet* is guaranteed to match the ordering of the calls on this *Variant*. The number of results will also be the same.

- **call_set_id** (*string*) – The ID of the call set this variant call belongs to.

  If this field is not present, the ordering of the call sets from a *SearchCallSetsRequest* over this *VariantSet* is guaranteed to match the ordering of the calls on this *Variant*. The number of results will also be the same.

- **genotype** (*ListValue*) – The genotype of this variant call.

  A 0 value represents the reference allele of the associated *Variant*. Any other value is a 1-based index into the alternate alleles of the associated *Variant*.

  If a variant had a referenceBases field of "T", an alternateBases value of ["A", "C"], and the genotype was [2, 1], that would mean the call represented the heterozygous value "CA" for this variant. If the genotype was instead [0, 1] the represented value would be "TA". Ordering of the genotype values is important if the phaseset field is present. Missing genotype genotypes may be indicated using the "dot annotation" [".", "."], as specified in VCF4.2; this is e.g. used for types of structural variants.

- **phaseset** (*string*) – If this field is populated, this variant call's genotype ordering implies the phase of the bases and is consistent with any other variant calls on the same contig which have the same phaseset string.

- **genotype_likelihood** (*double*) – The genotype likelihoods for this variant call. Each array entry represents how likely a specific genotype is for this call as log10(P(data | genotype)), analogous to the GL tag in the VCF spec. The value ordering is defined by the GL tag in the VCF spec.

- **attributes** (*Attributes*) – A map of additional information about the Call.

A *Call* represents the determination of genotype with respect to a particular *Variant*.

It may include associated information such as quality and phasing. For example, a call might assign a probability of 0.32 to the occurrence of a SNP named rs1234 in a call set with the name NA12345.

message Variant

Fields

- **id** (*string*) – The variant ID.

- **variant_set_id** (*string*) – The ID of the *VariantSet* this variant belongs to. This transitively defines the *ReferenceSet* against which the *Variant* is to be interpreted.

- **names** (*string*) – Names for the variant, for example a RefSNP ID.

- **created** (*long*) – The date this variant was created in milliseconds from the epoch.

- **updated** (*long*) – The time at which this variant was last updated in milliseconds from the epoch.

- **reference_name** (*string*) – The reference on which this variant occurs. (e.g. *chr20* or *X*)

- **start** (*long*) – The start position at which this variant occurs (0-based). This corresponds to the first base of the string of reference bases. Genomic positions are non-negative integers less than reference length. Variants spanning the join of circular genomes are represented as two variants one on each side of the join (position 0).

- **end** (*long*) – The end position (exclusive), resulting in [start, end) closed-open interval. This is typically calculated by *start* + *referenceBases.length*.

- **reference_bases** (*string*) – The reference bases for this variant. They start at the given start position.

- **alternate_bases** (*string*) – The bases that appear instead of the reference bases. Multiple alternate alleles are possible.
- **attributes** (*Attributes*) – A map of additional information about the Variant.
- **calls** (list of *Call*) – The variant calls for this particular variant. Each one represents the determination of genotype with respect to this variant. *Call's in this array are implicitly associated with this 'Variant*.
- **variant_type** (*string*) – The "variant_type" is used to denote e.g. structural variants. Examples: DUP : duplication of sequence following "start"; not necessarily in situ DEL : deletion of sequence following "start"
- **svlen** (*long*) – Length of the - if labeled as such in variant_type - structural variation. Based on the use in VCFv4.2
- **cipos** (*sint32*) – In the case of structural variants, start and end of the variant may not be known with an exact base position. "cipos" provides an interval with high confidence for the start position. The interval is provided by 0 or 2 signed integers which are added to the start position. Based on the use in VCFv4.2 Example: [-12000, 1000]
- **ciend** (*sint32*) – Similar to "cipos", but for the variant's end position (which is derived from start + svlen). Example: [-1000, 0]
- **filters_applied** (*boolean*) – True if filters were applied for this variant. VCF column 7 "FILTER" any value other than the missing value.
- **filters_passed** (*boolean*) – True if all filters for this variant passed. VCF column 7 "FILTER" value PASS.
- **filters_failed** (*string*) – Zero or more filters that failed for this variant. VCF column 7 "FILTER" shared across all alleles in the same VCF record.

A *Variant* represents a change in DNA sequence relative to some reference. For example, a variant could represent a SNP or an insertion. Variants belong to a *VariantSet*. This is equivalent to a row in VCF.

variant_service

SearchVariants(*request*)

Parameters **request** – *SearchVariantsRequest*

Return type *SearchVariantsResponse*

Throws *GAException*

Gets a list of *Variant* matching the search criteria.

POST /variants/search must accept a JSON version of *SearchVariantsRequest* as the post body and will return a JSON version of *SearchVariantsResponse*.

GetCallSet(*request*)

Parameters **request** – *GetCallSetRequest*

Return type *CallSet*

Throws *GAException*

Gets a *CallSet* by ID.

GET /callsets/{id} will return a JSON version of *CallSet*.

SearchVariantSets(*request*)

Parameters **request** – *SearchVariantSetsRequest*

Return type *SearchVariantSetsResponse*

Throws *GAException*

Gets a list of *VariantSet* matching the search criteria.

POST /variantsets/search must accept a JSON version of *SearchVariantSetsRequest* as the post body and will return a JSON version of *SearchVariantSetsResponse*.

GetVariantSet(*request*)

Parameters **request** – *GetVariantSetRequest*

Return type *VariantSet*

Throws *GAException*

Gets a *VariantSet* by ID.

GET /variantsets/{variant_set_id} will return a JSON version of *VariantSet*.

GetVariant(*request*)

Parameters **request** – *GetVariantRequest*

Return type *Variant*

Throws *GAException*

Gets a *Variant* by ID.

GET /variants/{id} will return a JSON version of *Variant*.

SearchCallSets(*request*)

Parameters **request** – *SearchCallSetsRequest*

Return type *SearchCallSetsResponse*

Throws *GAException*

Gets a list of call sets matching the search criteria.

POST /callsets/search must accept a JSON version of *SearchCallSetsRequest* as the post body and will return a JSON version of *SearchCallSetsResponse*.

message SearchVariantSetsRequest

Fields

- **dataset_id** (*string*) – The *Dataset* to search.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

********** /variantsets ********** This request maps to the body of *POST /variantsets/search* as JSON.

message SearchVariantSetsResponse

Fields

- **variant_sets** (list of *VariantSet*) – The list of matching variant sets.

- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /variantsets/search* expressed as JSON.

message GetVariantSetRequest

Fields

- **variant_set_id** (*string*) – The ID of the *VariantSet* to be retrieved.

This request maps to the URL *GET /variantsets/{id}*.

message SearchVariantsRequest

Fields

- **variant_set_id** (*string*) – The *VariantSet* to search.
- **call_set_ids** (*string*) – Only return variant calls which belong to call sets with these IDs. If unspecified, return all variants and no variant call objects.
- **reference_name** (*string*) – Required. Only return variants on this reference.
- **start** (*long*) – Required. The beginning of the window (0-based, inclusive) for which overlapping variants should be returned. Genomic positions are non-negative integers less than reference length. Requests spanning the join of circular genomes are represented as two requests one on each side of the join (position 0).
- **end** (*long*) – Required. The end of the window (0-based, exclusive) for which overlapping variants should be returned.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

********** /variants ********** This request maps to the body of *POST /variants/search* as JSON.

message SearchVariantsResponse

Fields

- **variants** (list of *Variant*) – The list of matching variants. If the *callSetId* field on the returned calls is not present, the ordering of the call sets from a *SearchCallSetsRequest* over the parent *VariantSet* is guaranteed to match the ordering of the calls on each *Variant*. The number of results will also be the same.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /variants/search* expressed as JSON.

message GetVariantRequest

Fields

- **variant_id** (*string*) – The ID of the *Variant* to be retrieved.

This request maps to the URL *GET /variants/{id}*.

message `SearchCallSetsRequest`

Fields

- **`variant_set_id`** (*`string`*) – The VariantSet to search.
- **`name`** (*`string`*) – Only return call sets with this name (case-sensitive, exact match).
- **`biosample_id`** (*`string`*) – Return only call sets generated from the provided Biosample ID.
- **`page_size`** (*`integer`*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **`page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

********** /callsets ********** This request maps to the body of *POST /callsets/search* as JSON.

message `SearchCallSetsResponse`

Fields

- **`call_sets`** (list of *`CallSet`*) – The list of matching call sets.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /callsets/search* expressed as JSON.

message `GetCallSetRequest`

Fields

- **`call_set_id`** (*`string`*) – The ID of the *CallSet* to be retrieved.

This request maps to the URL *GET /callsets/{call_set_id}*.

rna_quantification

Units for expression level. FPKM - number of Fragments Per Kilobase of feature length per Million reads FPKM is calculated by dividing the fragment count per feature by the total number of reads in millions (FPM - Fragments Per Million). FPM is then divided by feature length in kilobases to obtain FPKM. TPM - Transcripts per kilobase Per Million reads TPM is calculated by first dividing the fragment/read count by feature length in kilobases (RPK - Reads Per Kilobase). The count of all RPKs in the sample are then divided by a million to generate a 'per million' scaling value. For each feature RPK divided by the 'per million' scaling factor generated TPM.

enum `ExpressionUnit`

Symbols EXPRESSION_UNIT_UNSPECIFIED|FPKM|TPM

•*EXPRESSION_UNIT_UNSPECIFIED*:

•*FPKM*:

•*TPM*:

message `RnaQuantificationSet`

Fields

- **id** (*string*) – The RNA quantification set ID.
- **dataset_id** (*string*) – The ID of the dataset this RNA Quantification set belongs to.
- **name** (*string*) – The RNA quantification set name.
- **attributes** (*Attributes*) – A map of additional information about the Quantification Set.

A collection of associated RNAQuantifications. Typically this will be all the Quantifications of samples from an experiment. For example, a time course experiment would be described by a RnaQuantificationSet with the individual RNASeq experiments of the time point being represented as the member RnaQuantifications.

message RnaQuantification

Fields

- **id** (*string*) – The unique ID assigned to the results of running the described programs on the specified reads and assignment to the listed annotation.
- **name** (*string*) – Name
- **biosample_id** (*string*) – Biosample ID
- **description** (*string*) – Description
- **read_group_ids** (*string*) – ID(s) of the ReadGroup(s) providing the reads for the analysis.
- **programs** (list of *Program*) – Programs can be used to track the provenance of how read data was quantified.
- **feature_set_ids** (*string*) – List of annotation sets used.
- **rna_quantification_set_id** (*string*) – ID of the containing RnaQuantificationSet.
- **attributes** (*Attributes*) – A map of additional information about the Quantification.

Top level identifying information

message ExpressionLevel

Fields

- **id** (*string*) – Expression ID
- **name** (*string*) – Name
- **rna_quantification_id** (*string*) – The associated RnaQuantification
- **raw_read_count** (*float*) – The number of reads mapped to this feature.
- **expression** (*float*) – Numerical expression value.
- **is_normalized** (*boolean*) – True if the expression value is a normalized value.
- **units** (*ExpressionUnit*) – The units of the expression value if one is given.
- **score** (*float*) – Weighted score for the expression value.
- **conf_interval_low** (*float*) – Lower bound of the confidence interval on the expression value.
- **conf_interval_high** (*float*) – Upper bound of the confidence interval on the expression value.

- **attributes** (*Attributes*) – A map of additional information about the Expression Level.

The actual numerical quantification for each feature.

rna_quantification_service

Gets a list of 'ExpressionLevel' matching the search criteria.

This request maps to the URL *GET /rnaquantificationsets/{rna_quantification_set_id}*.

********** /rnaquantifications/search **********

This request maps to the URL *GET /rnaquantifications/{rna_quantification_id}*.

This request maps to the URL *GET /expressionlevels/{expression_level_id}*.

SearchRnaQuantifications(*request*)

Parameters **request** – *SearchRnaQuantificationsRequest*

Return type *SearchRnaQuantificationsResponse*

Throws *GAException*

Gets a list of 'RnaQuantification' matching the search criteria. 'POST /rnaquantifications/search' must accept JSON version of 'SearchRnaQuantificationsRequest' as the post body and will return a JSON version of 'SearchRnaQuantificationResponse'.

GetExpressionLevel(*request*)

Parameters **request** – *GetExpressionLevelRequest*

Return type *ExpressionLevel*

Throws *GAException*

Gets a *ExpressionLevel* by ID. *GET /expressionlevels/{id}* will return a JSON version of *ExpressionLevel*.

GetRnaQuantificationSet(*request*)

Parameters **request** – *GetRnaQuantificationSetRequest*

Return type *RnaQuantificationSet*

Throws *GAException*

Gets a *RnaQuantificationSet* by ID. *GET /rnaquantificationsets/{id}* will return a JSON version of *RnaQuantificationSet*.

SearchRnaQuantificationSets(*request*)

Parameters **request** – *SearchRnaQuantificationSetsRequest*

Return type *SearchRnaQuantificationSetsResponse*

Throws *GAException*

Gets a list of 'RnaQuantificationSet' matching the search criteria. 'POST /rnaquantificationsets/search' must accept JSON version of 'SearchRnaQuantificationSetRequest' as the post body and will return a JSON version of 'SearchRnaQuantificationSetResponse'.

SearchExpressionLevels(*request*)

Parameters **request** – *SearchExpressionLevelsRequest*

Return type *SearchExpressionLevelsResponse*

Throws *GAException*

'POST /expressionlevels/search' must accept JSON version of 'SearchExpressionLevelsRequest' as the post body and will return a JSON version of 'SearchExpressionLevelsResponse'.

GetRnaQuantification(*request*)

Parameters **request** – *GetRnaQuantificationRequest*

Return type *RnaQuantification*

Throws *GAException*

Gets a *RnaQuantification* by ID. *GET /rnaquantifications/{id}* will return a JSON version of *RnaQuantification*.

message SearchRnaQuantificationSetsRequest

Fields

- **dataset_id** (*string*) – The *Dataset* to search.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of 'nextPageToken' from the previous response.

This request maps to the body of 'POST /rnaquantificationsets/search' as JSON.

message SearchRnaQuantificationSetsResponse

Fields

- **rna_quantification_sets** (list of *RnaQuantificationSet*) – The list of matching quantification sets.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of 'nextPageToken' from the previous response.

This is the response from 'POST /rnaquantificationsets/search' expressed as JSON.

message GetRnaQuantificationSetRequest

Fields

- **rna_quantification_set_id** (*string*) – The ID of the *RnaQuantificationSet*.

message SearchRnaQuantificationsRequest

Fields

- **rna_quantification_set_id** (*string*) – Return only Rna Quantifications which belong to this set. Must be specified.
- **biosample_id** (*string*) – Return only RNA quantifications regarding the specified biosample. Optional.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of 'nextPageToken' from the previous response.

This request maps to the body of 'POST /rnaquantifications/search' as JSON.

message `SearchRnaQuantificationsResponse`

Fields

- **`rna_quantifications`** (list of *`RnaQuantification`*) – The list of matching quantifications.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of 'nextPageToken' from the previous response.

This is the response from 'POST /rnaquantifications/search' expressed as JSON.

message `GetRnaQuantificationRequest`

Fields

- **`rna_quantification_id`** (*`string`*) – The ID of the *RnaQuantification*.

message `SearchExpressionLevelsRequest`

Fields

- **`rna_quantification_id`** (*`string`*) – The rnaQuantification to restrict search to.
- **`names`** (*`string`*) – Only return expressions with any of the names (strict string matching).
- **`threshold`** (*`float`*) – Only return ExpressionLevel records with expressions exceeding this value. (Defaults to 0.0)
- **`page_size`** (*`integer`*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **`page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of 'nextPageToken' from the previous response.

********** /expressionlevels/search ********** This request maps to the body of 'POST /expressionlevels/search' as JSON.

message `SearchExpressionLevelsResponse`

Fields

- **`expression_levels`** (list of *`ExpressionLevel`*) – The list of matching quantifications.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of 'nextPageToken' from the previous response.

This is the response from 'POST /expressionlevels/search' expressed as JSON.

message `GetExpressionLevelRequest`

Fields

- **`expression_level_id`** (*`string`*) – The ID of the *ExpressionLevel*.

allele_annotations

The VariantAnnotation record groups different types of annotation records by Variant.

The TranscriptEffect sub record holds information on the effect of a specific allele on a specific transcript. As Variants may overlap multiple transcripts, they may have multiple TranscriptEffect records. Variants with multiple alternate

alleles will have multiple TranscriptEffect records per transcript. (2 alternate alleles x 3 transcripts = 6 TranscriptEffect records)

VariantAnnotation records belong to VariantAnnotationSets. VariantAnnotationSets are created by comparing a number of Variants from a VariantSet to a specific set of reference data using specific software tools. A VariantAnnotationSet contains information on reference data and software versions used in calculating the annotation; it is essential this information is exhaustive.

message `AnalysisResult`

Fields

- **`analysis_id`** (*string*) – The ID of the analysis record for this result
- **`result`** (*string*) – The text-based result for this analysis
- **`score`** (*integer*) – The numeric score for this analysis

An AnalysisResult record holds the output of a prediction package such as SIFT on a specific allele.

message `AlleleLocation`

Fields

- **`start`** (*integer*) – Relative start position of the allele in this coordinate system
- **`end`** (*integer*) – Relative end position of the allele in this coordinate system
- **`reference_sequence`** (*string*) – Reference sequence in feature (this should be the codon at CDS level)
- **`alternate_sequence`** (*string*) – Alternate sequence in feature (this should be the codon at CDS level)

An allele location record holds the location of an allele relative to a non - genomic coordinate system such as a CDS or protein and holds the reference and alternate sequence where appropriate

message `VariantAnnotationSet`

Fields

- **`id`** (*string*) – The ID of the variant annotation set record
- **`variant_set_id`** (*string*) – The ID of the variant set to which this annotation set belongs
- **`name`** (*string*) – The variant annotation set name.
- **`analysis`** (*Analysis*) – Analysis details. It is essential to supply versions for all software and reference data used.
- **`attributes`** (*Attributes*) – A map of additional information about the Annotation Set.

A VariantAnnotationSet record groups VariantAnnotation records. It is derived from a VariantSet and holds information describing the software and reference data used in the annotation.

message `HGVSAnnotation`

Fields

- **`genomic`** (*string*) –
- **`transcript`** (*string*) –
- **`protein`** (*string*) –

A HGVSAnnotation record holds Human Genome Variation Society descriptions of the sequence change with respect to genomic, transcript and protein sequences. See: http:// www.hgvs.org/mutnomen/recs.html. Descriptions should be provided at genomic level - Descriptions at transcript level should be provided when the allele lies within a transcript - Descriptions at protein level should be provided when the allele lies within the translated sequence or stop codon.

message `TranscriptEffect`

Fields

- **`id`** (*string*) – The ID of the transcript effect record
- **`feature_id`** (*string*) – The id of the transcript feature the annotation is relative to. TODO: derive unique id from digest of data [location, allele, transcript?]
- **`alternate_bases`** (*string*) – Alternate allele - a variant may have more than one alternate allele, each of which will have distinct annotation.
- **`effects`** (list of *OntologyTerm*) – Effect of variant on this feature.
- **`hgvs_annotation`** (*HGVSAnnotation*) – Human Genome Variation Society variant descriptions.
- **`cdna_location`** (*AlleleLocation*) – Change relative to cDNA.
- **`cds_location`** (*AlleleLocation*) – Change relative to coding sequence.
- **`protein_location`** (*AlleleLocation*) – Change relative to protein.
- **`analysis_results`** (list of *AnalysisResult*) – Output from prediction packages such as SIFT.
- **`attributes`** (*Attributes*) – A map of additional information about the Transcript Effect.

A transcript effect record is a set of information describing the effect of an allele on a transcript

message `VariantAnnotation`

Fields

- **`id`** (*string*) – The ID of this VariantAnnotation.
- **`variant_id`** (*string*) – The variant ID.
- **`variant_annotation_set_id`** (*string*) – The ID of the variant annotation set this record belongs to.
- **`created`** (*string*) – The time at which this record was created, in ISO 8601 format.
- **`transcript_effects`** (list of *TranscriptEffect*) – The transcript effect annotation for the alleles of this variant. Each one represents the effect of a single allele on a single transcript.
- **`attributes`** (*Attributes*) – A map of additional information about the Annotation.

A *VariantAnnotation* record represents the result of comparing a variant to a set of reference data.

allele_annotation_service

********** **/variantannotations** ********** This section will be re-instated when features are available in the API

Only return variant annotations for any of these features. Features may include specific transcripts or genes. A search by gene will return information for all transcripts associated with the gene in the variant annotation set.

This or a location (referenceName/referenceId plus optional start and end) must be supplied. If empty, return all variant annotations in specified window. repeated string feature_ids;

********** /variantannotationsets **********

SearchVariantAnnotationSets(*request*)

Parameters **request** – *SearchVariantAnnotationSetsRequest*

Return type *SearchVariantAnnotationSetsResponse*

Throws *GAException*

Returns a list of available variant annotation sets.

POST /variantannotationsets/search must accept a JSON version of *SearchVariantAnnotationSetsRequest* as the post body and will return a JSON version of *SearchVariantAnnotationSetsResponse*.

SearchVariantAnnotations(*request*)

Parameters **request** – *SearchVariantAnnotationsRequest*

Return type *SearchVariantAnnotationsResponse*

Throws *GAException*

Gets a list of *VariantAnnotation* messages matching the search criteria.

This allows the mining of allele-specific annotations on a VariantSet by either a region or by a set of genomic features. Where a region is supplied annotation of all alleles vs all features in the region is returned. Where a set of features is supplied, only annotations against these features (matching on featuretype and id) are returned and other overlapping features are ignored.

variantannotationsets/search returns information on the input to the annotation. This will be a VariantSet and the reference data and software versions used in calculating the annotation. It is essential this information is exhaustive.

POST /variantannotations/search must accept a JSON version of *SearchVariantAnnotationsRequest* as the post body and will return a JSON version of *SearchVariantAnnotationsResponse*.

GetVariantAnnotationSet(*request*)

Parameters **request** – *GetVariantAnnotationSetRequest*

Return type *VariantAnnotationSet*

Throws *GAException*

Gets an *VariantAnnotationSet* by ID.

GET /variantannotationsets/{variant_annotation_set_id} will return a JSON version of *VariantAnnotationSet*.

message SearchVariantAnnotationsRequest

Fields

- **variant_annotation_set_id** (*string*) – Required. The ID of the variant annotation set to search over.
- **reference_name** (*string*) – Only return variants with reference alleles on the reference with this name. One of this field or *reference_id* is required.
- **reference_id** (*string*) – Only return variants with reference alleles on the reference with this ID. One of this field or *reference_name* is required.
- **start** (*long*) – Required if reference_name or reference_id supplied. The beginning of the window (0-based, inclusive) for which variants with overlapping reference alleles should be returned. Genomic positions are non-negative integers less than reference length.

Requests spanning the join of circular genomes are represented as two requests one on each side of the join (position 0).

- **end** (*long*) – Required if reference_name or reference_id supplied. The end of the window (0-based, exclusive) for which variants with overlapping reference alleles should be returned.
- **effects** (list of *OntologyTerm*) – This filter allows variant, transcript combinations to be extracted by effect type(s). Only return variant annotations including any of these effects and only return transcript effects including any of these effects. Exact matching across all fields of the Sequence Ontology OntologyTerm is required. (A transcript effect may have multiple SO effects which will all be reported.) If empty, return all variant annotations.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request maps to the body of *POST /variantannotations/search* as JSON.

message SearchVariantAnnotationsResponse

Fields

- **variant_annotations** (list of *VariantAnnotation*) – The list of matching variant annotations.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /variantannotations/search* expressed as JSON.

message SearchVariantAnnotationSetsRequest

Fields

- **variant_set_id** (*string*) – Required. The *VariantSet* to search.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request maps to the body of *POST /variantannotationsets/search* as JSON.

message SearchVariantAnnotationSetsResponse

Fields

- **variant_annotation_sets** (list of *VariantAnnotationSet*) – The list of matching variant annotation sets.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /variantannotationsets/search* expressed as JSON.

message GetVariantAnnotationSetRequest

Fields

- **variant_annotation_set_id** (*string*) – The ID of the *VariantAnnotationSet* to be retrieved.

This request maps to the URL *GET /variantannotationsets/{id}*.

sequence_annotations

This protocol defines annotations on GA4GH genomic sequences It includes two types of annotations: continuous and discrete hierarchical.

The discrete hierarchical annotations (called Features) are derived from the Sequence Ontology (SO) and GFF3 work

http://www.sequenceontology.org/gff3.shtml

The goal is to be able to store annotations using the GFF3 and SO conceptual model, although there is not necessarly a one-to-one mapping in Protobuf messages to GFF3 records.

The minimum requirement is to be able to accurately represent the current state of the art annotation data and the full SO model. Feature is the core generic record which corresponds to the a GFF3 record.

The continuous data (called Continuous) represents numerical data associated with each base position, such as is stored in BigWig, Wiggle or BedGraph Formats

message Feature

Fields

- **id** (*string*) – Id of this annotation node.
- **name** (*string*) – An optional name to provide for the feature.
- **gene_symbol** (*string*) – The gene symbol the feature occurs on. This field may be replaced with a more generic representation in a future version.
- **parent_id** (*string*) – Parent Id of this node. Set to empty string if node has no parent.
- **child_ids** (*string*) – Ordered array of Child Ids of this node. Since not all child nodes are ordered by genomic coordinates, this can't always be reconstructed from parent_id's of the children alone.
- **feature_set_id** (*string*) – Identifier for the containing feature set.
- **reference_name** (*string*) – The reference on which this feature occurs (e.g. *chr20* or *X*).
- **start** (*long*) – The start position at which this feature occurs (0-based). This corresponds to the first base of the string of reference bases. Genomic positions are non-negative integers less than reference length. Features spanning the join of circular genomes are represented as two features one on each side of the join (position 0).
- **end** (*long*) – The end position (exclusive), resulting in [start, end) closed-open interval. This is typically calculated by *start* + *reference_bases.length*.
- **strand** (*Strand*) – The strand on which the feature is present.
- **feature_type** (*OntologyTerm*) – Feature that is annotated by this region. Normally, this will be a term in the Sequence Ontology.
- **attributes** (*Attributes*) – Name/value attributes of the annotation. Attribute names follow the GFF3 naming convention of reserved names starting with an upper cases character, and user-define names start with lower-case. Most GFF3 pre-defined attributes apply,

the exceptions are ID and Parent, which are defined as fields. Additional, the following attributes are added: * Score - the GFF3 score column * Phase - the GFF3 phase column for CDS features.

Node in the annotation graph that annotates a contiguous region of a sequence.

message `FeatureSet`

Fields

- **`id`** (*`string`*) – The ID of this annotation set.
- **`dataset_id`** (*`string`*) – The ID of the dataset this annotation set belongs to.
- **`reference_set_id`** (*`string`*) – The ID of the reference set which defines the coordinate-space for this set of annotations.
- **`name`** (*`string`*) – The display name for this annotation set.
- **`source_uri`** (*`string`*) – The source URI describing the file from which this annotation set was generated, if any.
- **`attributes`** (*`Attributes`*) – A map of additional Feature Set information.

A set of sequence features annotations.

message `Continuous`

Fields

- **`start`** (*`long`*) – The start position at which this signal occurs (0-based). This corresponds to the first base of the string of reference bases. Genomic positions are non-negative integers less than the reference length.
- **`values`** (*`double`*) – The contiguous data values. Unsampled bases are given as NaN.
- **`continuous_set_id`** (*`string`*) – Identifier for the containing continous set.
- **`reference_name`** (*`string`*) – The reference on which this signal is defined (e.g. *chr20* or *X*).

This message defines a format for exchanging continuous valued signal data, such as those produced experimentally (e.g. ChIP-Seq data) or through calculations (e.g. conservation scores). It can be used, for example, to share data from Wiggle, BigWig, and BedGraph sources.

message `ContinuousSet`

Fields

- **`id`** (*`string`*) – The ID of this annotation set.
- **`dataset_id`** (*`string`*) – The ID of the dataset this annotation set belongs to.
- **`reference_set_id`** (*`string`*) – The ID of the reference set which defines the coordinate-space for this set of annotations.
- **`name`** (*`string`*) – The display name for this annotation set.
- **`source_uri`** (*`string`*) – The source URI describing the file from which this annotation set was generated, if any.
- **`attributes`** (*`Attributes`*) – A map of additional Feature Set information.

A set of Continous messages. Continuous values can be sent as a single Continuous message containing all values or a series of Continuous messages to either limit the size of the values array or to skip NaN values.

sequence_annotation_service

********** /featuresets ********** ********** /features ********** ********** /continuoussets ********** ********** /continuous **********

GetFeature(*request*)

Parameters **request** – *GetFeatureRequest*

Return type *Feature*

Throws *GAException*

Gets a *Feature* by ID.

GET /features/{id} will return a JSON version of *Feature*.

SearchContinuousSets(*request*)

Parameters **request** – *SearchContinuousSetsRequest*

Return type *SearchContinuousSetsResponse*

Throws *GAException*

Gets a list of *ContinuousSet* matching the search criteria.

POST /continuoussets/search must accept a JSON version of *SearchContinuousSetsRequest* as the post body and will return a JSON version of *SearchContinuousSetsResponse*.

SearchContinuous(*request*)

Parameters **request** – *SearchContinuousRequest*

Return type *SearchContinuousResponse*

Throws *GAException*

Gets continuous values matching the search criteria.

POST /continuous/search must accept a JSON version of *SearchContinuousRequest* as the post body and will return a JSON version of *SearchContinuousResponse*.

GetFeatureSet(*request*)

Parameters **request** – *GetFeatureSetRequest*

Return type *FeatureSet*

Throws *GAException*

Gets a *FeatureSet* by ID.

GET /featuresets/{id} will return a JSON version of *FeatureSet*.

SearchFeatures(*request*)

Parameters **request** – *SearchFeaturesRequest*

Return type *SearchFeaturesResponse*

Throws *GAException*

Gets a list of *Feature* matching the search criteria.

POST /features/search must accept a JSON version of *SearchFeaturesRequest* as the post body and will return a JSON version of *SearchFeaturesResponse*.

SearchFeatureSets(*request*)

Parameters **request** – *SearchFeatureSetsRequest*

Return type *SearchFeatureSetsResponse*

Throws *GAException*

Gets a list of *FeatureSet* matching the search criteria.

POST /featuresets/search must accept a JSON version of *SearchFeatureSetsRequest* as the post body and will return a JSON version of *SearchFeatureSetsResponse*.

GetContinuousSet(*request*)

Parameters **request** – *GetContinuousSetRequest*

Return type *ContinuousSet*

Throws *GAException*

Gets a *ContinuousSet* by ID.

GET /continuoussets/{id} will return a JSON version of *ContinuousSet*.

message SearchFeatureSetsRequest

Fields

- **dataset_id** (*string*) – The *Dataset* to search.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

•This request maps to the body of *POST /featuresets/search* as JSON.

message SearchFeatureSetsResponse

Fields

- **feature_sets** (list of *FeatureSet*) – The list of matching feature sets.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /featuresets/search* expressed as JSON.

message GetFeatureSetRequest

Fields

- **feature_set_id** (*string*) – The ID of the *FeatureSet* to be retrieved.

This request maps to the URL *GET /featuresets/{id}*.

message SearchFeaturesRequest

Fields

- **feature_set_id** (*string*) – The annotation set to search within. Either *feature_set_id* or *parent_id* must be non-empty.
- **name** (*string*) – Only returns features with this name (case-sensitive, exact match).

- **gene_symbol** (*string*) – Only return features with matching the provided gene symbol (case-sensitive, exact match). This field may be replaced with a more generic representation in a future version.
- **parent_id** (*string*) – Restricts the search to direct children of the given parent *feature* ID. Either *feature_set_id* or *parent_id* must be non-empty.
- **reference_name** (*string*) – Only return features on the reference with this name (matched to literal reference name as imported from the GFF3).
- **start** (*long*) – Required, if name or symbol not provided. The beginning of the window (0-based, inclusive) for which overlapping features should be returned. Genomic positions are non-negative integers less than reference length. Requests spanning the join of circular genomes are represented as two requests one on each side of the join (position 0).
- **end** (*long*) – Required, if name or symbol not provided. The end of the window (0-based, exclusive) for which overlapping features should be returned.
- **feature_types** (*string*) – TODO: To be replaced with a fully featured ontology search once the Metadata definitions are rounded out. If specified, this query matches only annotations whose *feature_type* matches one of the provided ontology terms.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request maps to the body of *POST /features/search* as JSON.

message SearchFeaturesResponse

Fields

- **features** (list of *Feature*) – The list of matching annotations, sorted by start position. Annotations which share a start position are returned in a deterministic order.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /features/search* expressed as JSON.

message GetFeatureRequest

Fields

- **feature_id** (*string*) – The ID of the *Feature* to be retrieved.

This request maps to the URL *GET /features/{id}*.

message SearchContinuousSetsRequest

Fields

- **dataset_id** (*string*) – The *Dataset* to search.
- **page_size** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

•This request maps to the body of *POST /continuoussets/search* as JSON.

message `SearchContinuousSetsResponse`

Fields

- **`continuous_sets`** (list of *`ContinuousSet`*) – The list of matching feature sets.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /continuoussets/search* expressed as JSON.

message `GetContinuousSetRequest`

Fields

- **`continuous_set_id`** (*`string`*) – The ID of the *ContinuousSet* to be retrieved.

This request maps to the URL *GET /continuoussets/{id}*.

message `SearchContinuousRequest`

Fields

- **`continuous_set_id`** (*`string`*) – The annotation set to search within. Required value.
- **`reference_name`** (*`string`*) – Get continuous values on this reference. Required value.
- **`start`** (*`long`*) – The beginning of the window (0-based, inclusive) for which continuous values should be returned. Required value.
- **`end`** (*`long`*) – The end of the window (0-based, exclusive) for which continuous values should be returned. Required value.
- **`page_size`** (*`integer`*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **`page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request maps to the body of *POST /continuous/search* as JSON.

message `SearchContinuousResponse`

Fields

- **`continuous`** (list of *`Continuous`*) – The list of matching continuous values, sorted by start position. All sampled values within the query range are returned. Unsampled values are assigned 'NaN' value. The values returned do not necessarily cover the same range as the query as all unsampled values might not be returned or if the query range extends beyond the reference range.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /continuous/search* expressed as JSON.

struct

enum `NullValue`

Symbols NULL_VALUE

NullValue is a singleton enumeration to represent the null value for the *Value* type union.

The JSON representation for *NullValue* is JSON *null*.

•*NULL_VALUE*: Null value.

message `Struct`

Fields

- **`fields`** (map< string , *`Value`* >) – Unordered map of dynamically typed values.

Struct represents a structured data value, consisting of fields which map to dynamically typed values. In some languages, *Struct* might be supported by a native representation. For example, in scripting languages like JS a struct is represented as an object. The details of that representation are described together with the proto support for the language.

The JSON representation for *Struct* is JSON object.

message `Value`

Fields

- **`kind`** (*`NullValue`* | double | string | boolean | *`Struct`* | *`ListValue`*) – *`NullValue`*: Represents a null value.

  double: Represents a double value.

string: Represents a string value.

boolean: Represents a boolean value.

Struct: Represents a structured value.

ListValue: Represents a repeated *Value*.

Value represents a dynamically typed value which can be either null, a number, a string, a boolean, a recursive struct value, or a list of values. A producer of value is expected to set one of that variants, absence of any variant indicates an error.

The JSON representation for *Value* is JSON value.

message `ListValue`

Fields

- **`values`** (list of *Value*) – Repeated field of dynamically typed values.

ListValue is a wrapper around a repeated field of values.

The JSON representation for *ListValue* is JSON array.

genotype_phenotype

This protocol defines the associations between genotype and phenotype (G2P). Associations can be made as a result of literature curation, computational modeling, inference, etc., and modeled and shared using this schema.

Here, we follow the dogma of: Genotype + Environment = Phenotype

Where a G2P association is between the G(enotype) in the context of some E(nvironment), which gives rise to a P(henotype). These associations have further evidence, provenance, and attribution. We leverage the GenomicFeature in the sequenceAnnotation schema here as it can accomodate any genomic feature from a single nucleotide variation (SNV), up through a gene, and/or complex rearrangements. Each can be modeled as genomic features, and generally linked to a phenotype. Collections of these features can represent a genotype at different levels of completeness. Therefore, we can represent single allelic variation, allelic complement, and multiple variants in a genotype that can each or collectively be associated with a phenotype. To enable standardized integration, this schema relies heavily on OntologyTerms, for typing phenotype, genomic features, and levels of evidence. Suggested ontologies to leverage include (with browser links):

- Human Phenotype Ontology (HPO): http://www.ontobee.org/browser/index.php?o=hp
- Disease Ontology (DO): http://purl.obolibrary.org/obo/DOID_4
- Sequence Ontology (SO): http://www.sequenceontology.org/browser/
- Evidence Code Ontology (ECO): http://www.ontobee.org/browser/index.php?o=ECO
- Phenotypic Qualities (PATO): http://www.ontobee.org/browser/index.php?o=PATO

message `PhenotypeAssociationSet`

Fields

- **`id`** (*string*) – The phenotype association set ID.
- **`name`** (*string*) – The phenotype association set name.
- **`dataset_id`** (*string*) – The ID of the dataset this phenotype association set belongs to.
- **`info`** (map< string , *ListValue* >) – Optional additional information for this phenotype association set.

The top level container for phenotype association data.

message `EnvironmentalContext`

Fields

- **`id`** (*`string`*) – The Environment ID.
- **`environment_type`** (*`OntologyTerm`*) – Examples of some environment types could be drawn from: Ontology for Biomedical Investigations (OBI): http://purl.obofoundry.org/obo/obi/browse Chemical Entities of Interest (ChEBI): http://www.ontobee.org/browser/index.php?o=chebi Environment Ontology (ENVO): http://www.ontobee.org/browser/index.php?o=ENVO Anatomy (Uberon): http://www.ontobee.org/browser/index.php?o=uberon
- **`description`** (*`string`*) – A textual description of the environment. This is used to complement the structured description in the environmentType field

The context in which a genotype gives rise to a phenotype. This is fairly open-ended; as a stub we have a simple ontology term. For example, a controlled term for a drug, or perhaps an instance of a complex environment including temperature and air quality, or perhaps the anatomical environment (gut vs tissue type vs whole organism).

message `PhenotypeInstance`

Fields

- **`id`** (*`string`*) – The Phenotype ID.
- **`type`** (*`OntologyTerm`*) – HPO is recommended
- **`qualifier`** (list of *`OntologyTerm`*) – PATO is recommended. Often this qualifier might be for abnormal/normal, or severity. For example, severe: http://purl.obolibrary.org/obo/PATO_0000396 or abnormal: http://purl.obolibrary.org/obo/PATO_0000460
- **`age_of_onset`** (*`OntologyTerm`*) – HPO is recommended, for example, subclasses of http://www.human-phenotype-ontology.org/hpoweb/showterm?id=HP:0011007 TODO: also allow quantitative recording?
- **`description`** (*`string`*) – A textual description of the phenotype. This is used to complement the structured phenotype description in the type field.
- **`info`** (map< string , *`ListValue`* >) – Additional annotation data in key-value pairs.

An association to a phenotype and related information. This record is intended primarily to be used in conjunction with variants, but the record can also be composed with other kinds of entities such as diseases

message `Evidence`

Fields

- **`evidence_type`** (*`OntologyTerm`*) – ECO or OBI is recommended
- **`description`** (*`string`*) – A textual description of the evidence. This is used to complement the structured description in the evidence_type field
- **`info`** (map< string , *`ListValue`* >) – Additional annotation data in key-value pairs.

Evidence for the phenotype association. This is also a stub for further expansion. We should consider moving this into it's own schema.

message `FeaturePhenotypeAssociation`

Fields

- **`id`** (*`string`*) – A unique identifier for the association.

- **phenotype_association_set_id** (*string*) – The ID of the PhenotypeAssociationSet this FeaturePhenotypeAssociation belongs to.
- **feature_ids** (*string*) – The set of features of the organism that bears the phenotype. This could be as complete as a full complement of variants, or as minimal as the confirmed variants that are known causation for the annotated phenotype. Examples of features could be variations at the nucleotide level, large rearrangements at the chromosome level, or relevant epigenetic markers. Relevant genomic feature types are suggested to be those typed in the Sequence Ontology (SO). The feature set can have only one item, and must not be null.
- **evidence** (list of *Evidence*) – The evidence for this specific instance of association between the features and the phenotype.
- **phenotype** (*PhenotypeInstance*) – The phenotypic component of this association.
- **description** (*string*) – A textual description of the association.
- **environmental_contexts** (list of *EnvironmentalContext*) – The context in which the phenotype arises. Multiple contexts can be specified - these are assumed to all hold together
- **info** (map< string , *ListValue* >) – Additional annotation data in key-value pairs.

An association between one or more genomic features and a phenotype. The instance of association allows us to link a feature to a phenotype, multiple times, each bearing potentially different levels of confidence, such as resulting from alternative experiments and analysis.

genotype_phenotype_service

********** /phenotypeassociationsets/search ********** ********** /phenotypes/search ********** ********** /featurephenotypeassociations/search **********

SearchPhenotypeAssociationSets(*request*)

Parameters **request** – *SearchPhenotypeAssociationSetsRequest*

Return type *SearchPhenotypeAssociationSetsResponse*

Throws *GAException*

Gets a list of association sets accessible through the API. *POST /phenotypeassociationsets/search* must accept a JSON version of *SearchPhenotypeAssociationSetsRequest* as the post body and will return a JSON version of *SearchPhenotypeAssociationSetsResponse*.

SearchPhenotype(*request*)

Parameters **request** – *SearchPhenotypesRequest*

Return type *SearchPhenotypesResponse*

Throws *GAException*

Gets a list of phenotypes accessible through the API. *POST /phenotypes/search* must accept a JSON version of *SearchPhenotypesRequest* as the post body and will return a JSON version of *SearchPhenotypeAssociationSetsResponse*.

SearchPhenotypeAssociations(*request*)

Parameters **request** – *SearchGenotypePhenotypeRequest*

Return type *SearchGenotypePhenotypeResponse*

Throws *GAException*

Gets a list of genotype-phenotype assoications accessible through the API. *POST /featurephenotypeassociations/search* must accept a JSON version of *SearchPhenotypesRequest* as the post body and will return a JSON version of *SearchPhenotypeAssociationSetsResponse*.

message `SearchPhenotypeAssociationSetsRequest`

Fields

- **`dataset_id`** (*`string`*) – The *Dataset* to search. Mandatory
- **`page_size`** (*`long`*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **`page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *nextPageToken* from the previous response.

This request maps to the body of *POST /phenotypeassociationsets/search* as JSON.

message `SearchPhenotypeAssociationSetsResponse`

Fields

- **`phenotype_association_sets`** (list of *`PhenotypeAssociationSet`*) – The list of matching phenotype association sets.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /phenotypeassociationsets/search* expressed as JSON.

message `OntologyTermQuery`

Fields

- **`terms`** (list of *`OntologyTerm`*) –

One or more ontology terms can be queried together.

message `ExternalIdentifierQuery`

Fields

- **`ids`** (list of *`ExternalIdentifier`*) –

One or more ids can be queried together. Generally used for instances of a particular class of object (e.g. a specific gene or SNP).

message `EvidenceQuery`

Fields

- **`evidenceType`** (*`OntologyTerm`*) – ECO or OBI is recommended
- **`description`** (*`string`*) – The system may support regex. https://www.w3.org/TR/xpath-functions/#regex-syntax
- **`external_identifiers`** (list of *`ExternalIdentifier`*) – Only match Evidence messages that have any of these external identifiers

Evidence for the phenotype association.

message `SearchPhenotypesRequest`

Fields

- **phenotype_association_set_id** (*string*) – The *PhenotypeAssociationSet* to search. Mandatory
- **id** (*string*) – Phenotype ID TODO remove if a get-by-phenotype ID endpoint is added
- **description** (*string*) – The system may support regex. https://www.w3.org/TR/xpath-functions/#regex-syntax
- **type** (*OntologyTerm*) – Only return results that match this type
- **qualifiers** (list of *OntologyTerm*) – terms should be OR'd together. e.g. (severe OR abnormal)
- **age_of_onset** (*OntologyTerm*) – Only return results that match this age of onset
- **page_size** (*long*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

message SearchPhenotypesResponse

Fields

- **phenotypes** (list of *PhenotypeInstance*) – The list of matching PhenotypeInstances.
- **next_page_token** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /phenotypes/search* expressed as JSON.

message SearchGenotypePhenotypeRequest

Fields

- **phenotype_association_set_id** (*string*) – The *PhenotypeAssociationSet* to search. Mandatory
- **feature_ids** (*string*) – At least one feature_id or phenotype_id must be provided.
- **phenotype_ids** (*string*) – Phenotype IDs
- **evidence** (list of *EvidenceQuery*) – evidence
- **page_size** (*long*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **page_token** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *nextPageToken* from the previous response.

This request maps to the body of *POST /featurephenotypeassociations/search* as JSON.

The goal here is to allow users to query using one or more of Genotype, Phenotype, Environment, and Evidence.

A query using one of the above items is to mean, by default, that the remainder of the query is as a "wildcard", such that all matches to just that query term would come back. Combinations of the above are to act like AND rather than OR.

The "genotype" part of the query methods can be one or more genomic features. Associations can be made at many levels of granularity (from whole genotypes down to individual SNVs), but users may use these methods with partial or inexact information. Therefore, the query methods must be able to support query of some or

all of the associated features. Furthermore, use of the relationships between genomic features means that when querying for a gene, any variants to that gene are also returned. For example, a query with BRCA2 would mean that in addition to any direct associations to the BRCA2, all associations to sequence variants of BRCA2 would also be returned. Similarly, queries with OntologyTerms should perform the subclass closure.

Each query can be made against a string, an array of external identifers (such as for gene or SNP ids), ontology term ids, or full feature/phenotype/evidence objects.

message `SearchGenotypePhenotypeResponse`

Fields

- **`associations`** (list of *`FeaturePhenotypeAssociation`*) – The list of matching FeaturePhenotypeAssociation.
- **`next_page_token`** (*`string`*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /genotypephenotype/search* expressed as JSON.

peer_service

This interface provides a peer listing functionality for servers. Clients can create listings of GA4GH services by requesting peer lists from known network participants.

Network membership is voluntary, in that each peer is free to manage its peer list.

To announce a service to a peer send an AnnouncePeerRequest to a known good peer to the */announce* endpoint.

To receive information about a peer request a GetInfoResponse from the */info* endpoint.

To get a list of known peers send a ListPeersRequest to the */peers/list* endpoint.

`Info`(*request*)

Parameters **`request`** – *`GetInfoRequest`*

Return type *`GetInfoResponse`*

Throws *`GAException`*

Provides peers with a way to identify the protocol version of a peer. Other information describing the service can be included in the *info* field of the *GetInfoResponse*.

GET /info will return a JSON version of *GetInfoResponse*. and does not require any parameters.

`AnnouncePeer`(*request*)

Parameters **`request`** – *`AnnouncePeerRequest`*

Return type *`AnnouncePeerResponse`*

Throws *`GAException`*

Allows a client to notify a service of a potential peer. Services are expected to log these requests and implement policies for adding peers to their peer lists as desired.

The AnnouncePeerResponse only notifies the requester whether the request was valid. To find if their announce request has been accepted they must make a *ListPeersRequest*. *POST /peers/announce* must accept a JSON version of *AnnouncePeerRequest* as the post body and will return a JSON version of *AnnouncePeerResponse*.

`ListPeers`(*request*)

Parameters **`request`** – *`ListPeersRequest`*

Return type *ListPeersResponse*

Throws *GAException*

Gets a list of *Peer* messages that are being managed by the service. If the peer list becomes very long it is spread across multiple pages.

POST /peers/list must accept a JSON version of *ListPeersRequest* as the post body and will return a JSON version of *ListPeersResponse*.

message `ListPeersRequest`

Fields

- **`page_size`** (*integer*) – Specifies the maximum number of results to return in a single page. If unspecified, a system default will be used.
- **`page_token`** (*string*) – The continuation token, which is used to page through large result sets. To get the next page of results, set this parameter to the value of *next_page_token* from the previous response.

This request maps to the body of *POST /peers/list*.

message `ListPeersResponse`

Fields

- **`peers`** (list of *Peer*) – The list of *Peer* messages presented by the server. They are not expected in any particular order.
- **`next_page_token`** (*string*) – The continuation token, which is used to page through large result sets. Provide this value in a subsequent request to return the next page of results. This field will be empty if there aren't any additional results.

This is the response from *POST /peers/list* expressed as JSON.

message `AnnouncePeerRequest`

Fields

- **`peer`** (*Peer*) – This message contains information that can be used to connect to a possible peer.

This is the request sent to *POST /peers/announce*.

message `AnnouncePeerResponse`

Fields

- **`success`** (*boolean*) – This message notifies the client whether the AnnouncePeerRequest was well formed.
- **`attributes`** (*Attributes*) – Other information regarding an AnnouncePeerRequest can be sent in the attributes field.

This is the response from *POST /peers/announce*.

message `GetInfoRequest`

message `GetInfoResponse`

Fields

- **`protocol_version`** (*string*) – The string of the protocol version offered by the service. For example, "0.6.0a10".

- **attributes** (*Attributes*) – A map of additional information about the service can be included.

This is the response from *GET /info*.

message Peer

Fields

- **url** (*string*) – This is the base URL where the service can be accessed from. It is expected to be fully formed and to include the port number if the port in use is not standard (http 80, https 443).

  For example, the peer at 1kgenomes would appear as: http://1kgenomes.ga4gh.org

  Trailing slashes should be ignored when constructing client requests based on this peer, and so shouldn't be included.

  This example shows a peer that has specified both a base path and a port. http://myapp.mycloudservice.com:8080/data/ga4gh

- **attributes** (*Attributes*) – A map of additional information about the service can be included.

Peers allow clients to represent services to each other so ad-hoc networks can be easily constructed.

Development

Release process

There are two types of releases: development releases, and stable bugfix releases. Development releases happen as a matter of course while we are working on a given minor version series, and may be either a result of some new features being ready for use or a minor bugfix. Stable bugfix releases occur when mainline development has moved on to another minor version, and a bugfix is required for the currently released version. These two cases are handled in different ways.

Development Python releases

Version numbers are MAJOR.MINOR.PATCH triples. Minor version increments happen when significant changes are required to the schema codebase, which will result in a significant departure from the previously released version, either in code layout or in functionality. During the normal process of development within a minor version series, patch updates are routinely and regularly released. (In some cases bugfix releases can also come with a suffix, e.g. `0.6.0a9.post1`.)

Making a release entails the following steps:

1. Create a PR against `master` that has the following changes:
   (a) update the release notes in `doc/source/changelog.rst` with a description of what is in the release
   (b) modify `python/requirements.txt` to pin the ga4gh-common package to a specific version
   (c) modify `python/constraints.txt` to comment out all the lines referencing ga4gh packages
   (d) modify `python/constraints.txt.default` to have the identical contents as `python/constraints.txt`
2. Once this has been merged, tag the release on GitHub (on the releases page) with the appropriate version number.
3. Fetch the tag from the upstream repo, and checkout this tag.

4. Create the distribution tarball using `python setup.py sdist`, and then upload the resulting tarball to PyPI using `twine upload dist/ga4gh-schemas-$MAJOR.$MINOR.$PATCH.tar.gz` (using the correct file name).
5. Verify that the documentation at https://readthedocs.org/projects/ga4gh-schemas/en/stable/ is for the correct version (it may take a few minutes for this to happen after the release has been tagged on GitHub). The release notes docs should have changed, so that is a good section to look at to confirm the change.

All of the above steps after the tag is dropped on the target commit are now automated using Travis' capability to deploy to Pypi.

When releasing the package to PyPi, the release manager should guarantee the protocol version in `setup.py` matches the protocol version being released.

Since PyPi and the source code may provide different minor versions, release notes for the PyPi package are maintained in `python/README.rst`.

Stable bugfix release

When a minor version series has ended because of some significant shift in the server internals, there will be a period when the `master` branch is not in a releasable state. If a bugfix release is required during this period, we create a release using the following process:

1. If it does not already exist, create a release branch called `release-$MAJOR.$MINOR` from the tag of the last release.
2. Fix the bug by either cherry picking the relevant commits from `master`, or creating PRs against the `release-$MAJOR.$MINOR` branch if the bug does not apply to `master`.
3. Follow steps 1-5 in the process for **'Development releases'_** above, except using the `release-$MAJOR.$MINOR` branch as the base instead of `master`.

Changelog

Schema Release `v0.6.0a10`

Changes to `ga4gh/schemas` `master` branch since version `0.6.0a9` (Jan 23, 2016)

Features:

- Remove feature_id from ExpressionLevel and added ability to search by Name field. #818 Impacts
- `POST /expressionlevels/search`
- `GET /expressionlevels/{id}`
- Added support for BigWig files in a new Continuous Data object #802 Adds the following new endpoints:
- Continuous Data: `POST /continuoussets/search`
- Continuous Data: `GET /continuoussets/{id}`
- Continuous Data: `POST /continuous/search`
- Add deep set/get attr to protocol module #816
- Changed ontology term "id" to "term_id" #805 Impacts the message type OntologyTerm
- Replaced info fields with rich type Attributes fields #700 Impacts the following message types:
- TranscriptEffect

- VariantAnnotation
- Individual
- Biosample
- Experiment (new)
- Analysis (new)
- Dataset
- ReadGroup
- ReadGroupSet
- ReadAlignment
- Reference
- ReferenceSet
- RnaQuantificationSet
- RnaQuantification
- ExpressionLevel
- Feature
- VariantSetMetadata
- CallSet
- Call
- Variant
- Add ability to list and join peer server networks #760 Adds the following new endpoints:
- Peer: `POST /peers/list`
- Peer: `POST /peers/announce`
- Peer: `GET /info`
- Replace NCBI taxon ID integer with ontology term #699 Impacts the following message types:
- Reference
- ReferenceSet

Documentation:

- Add instructions for viewing the UML diagram #835
- Expand on schema usage instructions #786
- Expand on release process documentation #753

Infrastructure:

- Automatically deploy tagged releases to PyPi from Travis #825
- Add tests to verify constraints files #817
- Continued effort to separate out common methods

Schema Release v0.6.0a9

Changes to ga4gh/schemas master branch since version 0.6.0a8 (Oct 26, 2016)

- Fixed bugs: * Fixed typo in get: /v0.6.0a8/variantannotationset/{variant_annotation_set_id} *
- Fix to be able to handle VCFs with genotype == ./.
- Upgrade to use protobuf release 3.1
- Introduced a pip installable schemas package called ga4gh-schemas. We have also created pip installable packages for a support library called ga4gh-common and a client library module called ga4gh-client.
- Introduced a schemas package release to Maven Central. We will be posting regular ga4gh packages for each official schema release to Maven going forward.
- Changed the name of the biosample terms to track be consistent with the use of camel-case and the underscore character.
- Added a new schema visualization tool to create UML diagrams from the schemas. The new diagrams can be viewed on the Schemas page in the Read The Docs documents.

Schema Release v0.6.0a8

Changes to ga4gh/schemas master branch since version 0.6.0a7 (Aug 19, 2016)

- Introduced G2P API endpoints including the following:
- POST /phenotypeassociationsets/search
- POST /phenotypes/search
- POST /featurephenotypeassociations/search
- Add biometadata to RNA quantifications
- Add protobuf based HTTP annotations

Schema Release v0.6.0a7

Changes to ga4gh/schemas master branch since version 0.6.0a6 (Jul 25, 2016)

Introduced RNA API endpoints including the following:

- POST /rnaquantificationsets/search
- GET /rnaquantificationsets/{id}
- POST /rnaquantifications/search
- GET /rnaquantifications/{id}
- POST /expressionlevels/search
- GET /expressionlevels/{id}

Schema Release v0.6.0a6

Changes to ga4gh/schemas master branch since version 0.6.0a5 (Jun 20, 2016) NOTE: release notes have not been updated for several versions.

- Metadata section added
- Now support searching features by 'name' and 'gene_symbol'

Schema Release v0.6.0a5

Changes to ga4gh/schemas master branch since version v0.6.0a4 (Apr 7, 2016)

First Protocol Buffers (protobuf v3.0.0) version of the GA4GH API. Same set of features (messages, endpoints) as previous alpha release.

IMPORTANT: The switch from AVRO to protobuf in this pre-release will break compatibility with all client applications written against the previous AVRO schema version.

Schema Release v0.6.0a4

Changes to ga4gh/schemas master branch since version v0.6.0a3 (Mar 1, 2016)

Introduced Sequence Annotations API record types (FeatureSet and Feature) and associated endpoints POST featuresets/search GET featuresets/<id> POST features/search GET features/<id>

Schema Release v0.6.0a3

Changes to ga4gh/schemas master branch since version v0.6.0a2 (Feb 24, 2016)

Changed properPlacement in Reads API to improperPlacement (defaults to False), corrected documentation on variation annotation and dateTime format.

Schema Release v0.6.0a2

Changes to ga4gh/schemas master branch since version v0.6.0a1 (Dec 15, 2015)

Added Variant Annotations draft API

Schema Release v0.6.0a1

Changes to ga4gh/schemas master branch since version v0.5.1 (Oct 2, 2014)

Pervasive changes

Renamed all protocols and records to remove the "GA" prefix from their names. (Hence GASearchVariantSetsRequest is now SearchVariantSetsRequest, etc.)
The exception is GAException, which is unchanged.

There is no longer one unitary namespace. Objects now reside in "`org.ga4gh.models`", methods in "`org.ga4gh.methods`".

Updated the version number to `0.6.0a1`.

Changes visible to API clients

Additions

`Dataset` is now utilized throughout the API.

New methods, HTTP endpoints:

- `getDataset` via GET `datasets/<id>`
- `searchDatasets` via POST `/datasets/search`
- `getReadGroupSet` via GET `/readgroupsets/<id>`
- `getReadGroup` via GET `/readgroups/<id>`
- `getVariant` via GET `/variants/<id>`
- `getVariantSet` via GET `/variantsets/<id>`
- `getCallSet` via GET `/callsets/<id>`

New and modified entities:

- New record type `ExternalIdentifier` introduced.
- New record type `ReadStats` now optionally part of a returned `ReadGroup` or `ReadGroupSet` object.
- `Position` record type now specifies strand via new enum `Strand` (in place of `boolean reverseStrand`).

Field default values:

- `SearchReadsRequest.start` as passed into `/reads/search` is optional and no longer defaults to 0.
- Boolean fields in `ReadAlignment` as returned from `/reads/search` now default to `null` instead of `false`.

Field types:

- `SearchVariantSetsRequest` as passed into `/variantsets/search` now takes a single dataset ID, not an array.
- `SearchCallSetsRequest` as passed into `/callsets/search` now takes a single variant set ID, not an array.
- `SearchReferenceSetsRequest` as passed into `/referencesets/search` changed two parameters from arrays to singletons:
  - `md5checksum`

  - accession

New fields:

- SearchReferencesRequest passed into /references/search now accepts a referenceSetId parameter.
- SearchVariantsRequest passed into /variants/search now takes a variantSetId parameter.
- SearchReadsRequest passed into /reads/search now includes readGroupIds.
- SearchReadGroupSetsRequest passed into /readgroupsets/search takes a datasetId.
- ReferenceSet returned from /references/search now includes a name.
- VariantSet returned from /variants/search adds name and reference set ID fields.
- ReadGroup returned from /readgroups/<id> adds stats.

Changes internal to Schemas, documentation and organization

Removed

- src/main/resources/avro/beacon.avdl
- Files designated "Work In Progress" ("wip"):
  - src/main/resources/avro/wip/metadata.avdl
  - src/main/resources/avro/wip/metadatamethods.avdl
  - src/main/resources/avro/wip/variationReference.avdl

Changed

Moved GAException to methods.avdl.

Documentation

Using Doxygen to generate HTML documentation from schema (*.avdl) files.

Clarifications:

- How the SearchReadGroupSetsRequest.name field is interpreted.
- The meaning of SearchCallSetsRequest.name.
- Reference, ReferenceSet docs.

Updated CONTRIBUTING.md to describe the latest contribution rules.

Moved GeneratingDocumentation.md to doc/.

README.md now includes information about the Metadata Task Team.

Tests

Added tests to ensure Maven processes the schemas into a `jar` file successfully, and that we can compile the schemas into Python.

Appendix

Glossary

A compendium of bioinformatics terminology commonly used across the API.

Short Reads and BAM

High throughput genome and transcriptome sequencing produces millions of short (50-200 nucleotide) sequences. These sequences are usually referred to as reads. Reads can be produced from:

1. Complete genomes. These reads can be used to piece together the full genome of an individual.
2. Exomes. These reads are derived from just the gene regions in the genome (in humans this is a reduction of >97%)
3. Transcriptomes. Here, the RNA that gets transcribed from the genomic DNA is sequenced, representing only the genes that are active in the tissue that was sampled. Transcriptomes differ from tissue to tissue and can be used to determine differences between tumors and their surrounding tissues.

Reads are usually mapped to a reference genome, for example GRCh37 in humans.

These alignments can be displayed like so:

```
ID      CHROM  POS    CIGAR   SEQUENCE
read1   chr1   234    10M     GACAGTCCCA
read2   chr14  1456   10M     AAAGATTGAC
read3   chrX   2837   7M2I1M  TGGGACTCTA
```

In this format, the CIGAR string shows how well the read matches the genome: read1 is identical to the genome sequence over all its 10 bases: 10M. The first seven bases of read3 match the genome (7M), but then it has a 2 base insertion (2I), followed by another 1 base match (1M).

The SAM/BAM Format is a way of representing read data. It includes the fields shown above as well as information on read orientation, sequence quality, and optional fields. The format also allows for reads that do not align to the genome, by leaving the reference sequence ID, position, and cigar fields empty.

SAM is a human readable format, BAM is a condensed binary format. The formats can be readily converted to each other.

Genetic variants and VCF

Genetic variants are differences in the genome sequence from one individual to the next. Such variation can manifest at different scales, from small changes affecting just one or a few DNA base pairs, to copy number variations of whole exons or genes, to large structural variations affecting megabases or more. The GA4GH Variants schema focuses on small variants for now, because there's less consensus on how to represent the larger kinds of variation.

Small genetic variants can be represented as edits to a Reference sequence: typically a tuple of (1) Reference sequence name, (2) starting position of the affected portion on the Reference sequence, (3) DNA sequence of the affected portion

of the Reference, and (4) alternate DNA sequence found in place of the Reference sequence. Both the reference and alternate sequences are provided in order to represent sequence insertions and deletions (indels). A few examples:

CHROM	POS	REF	ALT
20	14370	G	A
20	17330	TA	T
20	18302	TG	ACC

The first variant is a single-nucleotide substitution of G to A. The second variant is a deletion of an A at position 17,331. The third variant is a multi-nucleotide change to a lengthier sequence starting at position 18,302.

Given a list of such variants, we can specify the genotype of one or more individuals with respect to each variant. The genotype of a diploid individual (for an autosomal variant) may take one of three distinct values: homozygous reference (0,0), heterozygous (0,1), or homozygous alternate (1,1). We can then present a matrix of genotypes, where the rows are variants as shown above, the columns are the individuals, and each entry is one of those three genotype *calls* (or marked missing):

CHROM	POS	REF	ALT	Alice	Bob
20	14370	G	A	(0,0)	(0,1)
20	17330	TA	T	(0,0)	(1,1)
20	18302	TG	ACC	(0,1)	–

(If the phase of an individual's genotypes across several variant positions is known, then the heterozygous genotypes (0,1) and (1,0) may be considered distinct, where the order specifies which homologous chromosome possesses the alternate sequence.)

It's possible to observe multiple different alternate sequences, or *alleles*, affecting the same portion of the reference. This can occur even within one individual, if their two homologous chromosomes contain different alternate sequences, and becomes somewhat common when representing variants observed across a population. To handle these cases, we allow a variant to specify multiple alternate alleles. For example:

CHROM	POS	REF	ALT
20	19254	G	A,C,T
20	21672	AT	AC,TGA

And in this case the genotypes can take values such as (0,3) or (1,2). This multi-allelic sites model was refined and popularized in the 1000 Genomes Project's Variant Call Format (VCF), upon which the Variants schema is based.

There remain some outstanding challenges with this model of small variants. For example, the same edit to the reference sequence can be represented in multiple ways. There are also different ways to represent clusters of alleles that affect overlapping but non-equal portions of the reference. The GA4GH doesn't yet prescribe resolutions to these ambiguities, and different conventions are used in practice.

(TODO possible additional/advanced topics: homozygous ref vs. no-call; phasing and phase sets; genotype likelihoods; INFO, FORMAT, QUAL, FILTER)

Use of Ontologies in GA4GH API

Examples for OntologyTerm use

- Info: Ontogenesis blog
- Info: Working implementation of the GA4GH docsystem's Ontologies document

Why should we use an ontology term?

- Info: http://ontogenesis.knowledgeblog.org/1296

A user may want to retrieve the rsIDs of all genomic variants to ciliopathies. Each rsID is annotated with a specific disease (e.g. Bardet Biedl syndrome, orofaciodigital syndrome). To query by a the functional grouping 'ciliopathy', classification of these diseases ciliopathies is needed, and can be provided through an ontology.

Ontology Lookup Service

- http://www.ebi.ac.uk/efo/EFO_0003900
- multiple child classes are returned, including those without a lexical match to the disease name

The effective use of ontology lookups requires the annotation of rsIDs with unique identifiers for the associated diseases, so that a programmatic lookup can use these to identify their parents and/or relations. Text queries are likely to return partial or erroneous result sets. Ontologies overlap in their scope, design and content. In the case of results from different ontologies, which may have a varying depth, the executioner of the query has to judge about the optimal scope of the returned data.

What is the minimum attribute requirement for OntologyTerm in GA4GH?

Conceptually (and consistent with the metadata branch)

termId required and implemented as CURIE we assume this resolves to a meaningful document, e.g. http://purl.obolibrary.org/obo/SO_0000147, using a prefix mapper, e.g. SO: <=> http://purl.obolibrary.org/obo/**SO**_

term preferred but not required (e.g. 'exon'); corresponds to class label

Ontology Selection and Overlap

Sometimes a single ontology provides excellent coverage of a domain. For example the Sequence Ontology is used successfully in GFF files to annotate exons, introns etc. There are multiple ontologies in some domains which overlap in scope and content and also interoperation between ontologies. For example the Human Phenotype Ontology (HP) provides terms describing human phenotypes. Disease phenotype associations are not provided in the HP, rather as supporting files with common cross references such as OMIM identifiers. When selecting an ontology consider coverage - how much of your data is represented; structure - does the ontology provide structure that meets your use cases, e.g. contains a class ciliopathy (see above), update frequency, ability to request terms when needed, adherence to community standards e.g. OBO foundry provides recommendations on versioning strategy and term deprecation. Note that OBO policy dictates that when the meaning of a class changes, then the identifier/IRI is deprecated/obsoleted, and a new identifier/IRI is minted. As a consequence, many databases that store associations to OBO classes (genes, diseases to phenotype etc) do not record the version of the ontology, as the semantics of the ID can be treated as immutable.

- Info: Why use the Human Phenotype Ontology (blog post by Melissa Haendel)

Age, date, time interval values => ISO8601 (see *Date and Time*)

Examples

Genotypic sex

termId "PATO:0020001",

term "male genotypic sex" ,

Sequence Ontology

termId "SO:0001583",

term "missense_variant",

Human Phenotype ontology

termId "HP:0000819",

term "Diabetes mellitus",

---

termId "HP:0012059",

term "Lentigo maligna melanoma",

Body part (Uberon)

termId "UBERON:0003403",

term "skin of forearm",

Human disease ontology

termId "DOID:9351",

term "diabetes mellitus",

Experimental factor ontology

termId "EFO:0000400",

term "diabetes mellitus",

---

termId "EFO:0004422",

term "exome",

Unit Ontology

termId "UO:0000016",

term "millimetre",

The JSON Format

JSON, or JavaScript Object Notation, is officially defined here. It is the standard data interchange format for web APIs.

The GA4GH Web API uses a JSON wire protocol, exchanging JSON representations of the objects defined in its Protocol Buffers schemas. More information on the schemas is available in *Google Protocol Buffers*; basically, the Protocol Buffers type definitions say what attributes any given JSON object ought to have, and what ought to be stored in each of them.

GA4GH JSON Serialization

The GA4GH web APIs use Protocol Buffers IDL to define their schemas, and use the associated Google Protocol Buffers JSON serialization libraries. Notice that the Protocol Buffers IDL uses snake case, while the on-the-wire protocol is in camel case.

Serialization example

For example, here is the schema definition for Variants (with comments removed):

```
message Variant {
  string id = 1;
  string variant_set_id = 2;
  repeated string names = 3;
  int64 created = 4;
  int64 updated = 5;
  string reference_name = 6;
  int64 start = 7;
  int64 end = 8;
  string reference_bases = 9;
  repeated string alternate_bases = 10;
  Attributes attributes = 13;
  repeated Call calls = 12;
}
```

Here is a serialized variant in JSON. It's a bit of an edge case in some respects:

```
{
    "id": "gv79384-3200-11",
    "variantSetId": "vs-44-1",
    "names": [
        "rs110",
        "Victoria"
    ],
    "created": 1446842841,
    "updated": null,
    "start": 1000,
    "end": 1001,
    "referenceBases": "A",
    "alternateBases": [
        "C",
        "CTATCTT"
    ],
    "attributes": {
        "attr": {
            "numberOfPapers": {
```

```
                "values": [
                    {"int32Value": 11}
                ]
            },
            "variantFacts": {
                "values": [
                    {"stringValue": "is_long"},
                    {"stringValue": "is_interesting"}
                ]
            }
        }
    }
}
```

Things to notice:

- A serialized record contains no explicit information about its type.
- "repeated" types are serialized as JSON arrays.
- Maps are serialized as JSON objects.
- Messages are also serialized as JSON objects.
- Enums (not shown here) are serialized as JSON strings.

Google Protocol Buffers

What does the GA4GH web API take from Protocol Buffers?

The GA4GH web API uses the Google Protocol Buffers language and JSON serialization libraries.

The GA4GH web API presents a simple HTTP(S) and JSON interface to clients. It does **not** use Protocol Buffers's binary serialization format.

How does the GA4GH web API use Protocol Buffer schemas?

GA4GH web API objects, including both the data objects actually exchanged and the control messages requesting and returning those objects, are defined in Protocol Buffers.

The full documentation for the Protocol buffers language can be found here.

How does the GA4GH Web API use Protocol Buffers?

The GA4GH web API schemas are broken up into multiple proto files, which reference each other. Each file defines a number of message types, grouped into a "protocol" that defines a facet of the API. Mostly, the files come in pairs: a normal proto file defining the types representing actual data, and a "methods" proto file defining the control messages to be sent back and forth to query and exchange the representational types, and the URLs associated with various operations.

Each type has a leading comment documenting its purpose, and each field in the type has a description. These are included in the automatically generated API documentation.

Here is an example of a proto definition from , in this case defining a genomic *Position* type which is used across the API:

```
message Position {
  // The name of the `Reference` on which the `Position` is located.
  string reference_name = 1;

  // The 0-based offset from the start of the forward strand for that
  // `Reference`. Genomic positions are non-negative integers less than
  // `Reference` length.
  int64 position = 2;

  // Strand the position is associated with.
  Strand strand = 3;
}
```

This is a "message", which contains three fields. All of the fields are required to be filled in, and all of the fields can only hold objects of a particular single type. The last field holds a *Strand* object, which is defined elsewhere in the file.

Todo

- How much of the Protocol Buffers tutorial do we want in here?
- Document/show an example for methods (request and response pairing pattern)
- Talk about how we manually specify that some things land in URLs

---

Code samples (stub)

Useful tips from an online article about API development:

- Relevant information should be grouped together.
- Clarity is more important than efficiency or robustness.
- Simplicity is more important than a good-looking UI.

Notes on sample code:

- *DO* use hard-coded values to make the code easy to understand
- Variable, class, member and function names should be clear, feel free to use long names
- Forego exception handling, instead put in a comment indicating what kind of exceptions to handle in production code.

There is a long, useful section on sample code in the article listed above.

Notes on web APIs:

- There should be code samples in several languages
- Create Sample HTTP calls and JSON/ProtoBuf files

Samples should be followed by tables that describe each element as well as its data format. For example, it may not be enough to describe a parameter as a string. Are there special characters it can't handle? Are there limitations on its length? If an XML element is a date, you should specify the format of the date. If it's a time, then you need to specify its time zone.

Also, you'll need to explain how errors are handled. This may vary for the different formats that your API supports. If your API uses HTTP response codes to flag errors, these should be documented. Error documentation should explain why an error occurs and how to fix the problem.

Authentication is often required for Web APIs, and this needs to be documented in detail as well. If developers need API keys, be sure to give them step-by-step instructions on how to obtain these. Also, don't forget that Web APIs are built on top of HTTP, which is an incredibly rich protocol. You may have HTTP-related information that requires documentation, such as caching, content type and status codes.

Generating API Definitions

There has been an effort to standardize the methods for generating HTTP API descriptions that allow developers to rapidly develop gateways into their data. Since the GA4GH schemas are defined using Google Protocol Buffers IDL, it is possible to use this definition to generate documentation and code.

In this document we will generate swagger definitions for the GA4GH API using a plugin for the *protoc* compiler. For more on installing the protocol buffers compiler see INSTALL.rst.

Installing Prerequisites

Once you have the protocol buffers compiler installed, you'll need to install the *go* language bindings for your system. On Mac OS X this can be done using homebrew .

```
brew install go
```

Next create a directory that will contain the *go* packages we will install, and add it to your path. We will set that as our *GOPATH* and add it to the system *PATH*.

```
mkdir ~/golang
export GOPATH=~/golang
export PATH=$PATH:$GOPATH/bin
```

We install the required packages packages. For more information see: grpc-gateway

```
go get -u github.com/grpc-ecosystem/grpc-gateway/protoc-gen-grpc-gateway
go get -u github.com/grpc-ecosystem/grpc-gateway/protoc-gen-swagger
go get -u github.com/golang/protobuf/protoc-gen-go
```

Now that all of the prerequisites have been installed, we can generate swagger documents, which are JSON that describe the HTTP interface.

Compiling Swagger Documentation

Now we can use the *protoc* compiler with the addition of the plugin. First, we create the target directory. Then we run protoc with a few arguments.

The first argument tells the compiler to include any *proto* definitions in the current source tree when compiling. The second instructs the compiler to run the *swagger_out* plugin that will write to *target/swagger* . Lastly, we instruct the compiler to compile each proto file that ends in the same *service*.

```
$ mkdir -p target/swagger
$ protoc -Isrc/main/proto \
--swagger_out=logtostderr=true:target/swagger \
src/main/proto/ga4gh/*service.proto
```

This will create a directory (target/swagger/ga4gh) of JSON files describing the API that can be used with Open API Specification tools like swagger-codegen.

Using Generated Definitions

Swagger documents describe an HTTP interface and the messages it expects in a programmatic manner. This allows developers to use these generated documents to generate code and documentation. To quickly see what the generated documentation might look like, the contents of one the resulting JSON files can be pasted into the online editor at editor.swagger.io.

Using the online interface it is possible to export both client and server stubs in a number of languages. This service exposes the functionality of *swagger-codegen*, which we will install and provide an example of use.

swagger-codegen can be installed using homebrew on a Mac: *brew install swagger-codegen*. The available language bindings can be observed by running *swagger-codegen* from a terminal. Then, to generate a python client for the Read Service we can run:

```
$ swagger-codegen generate -i target/swagger/ga4gh/read_service.swagger.json -l␣
↪python -o ga4gh-reads-client
```

This will create a directory *ga4gh-reads-client* that includes most of the boilerplate, including README, *.gitignore*, etc., required to create a GA4GH client. This client can then be customized, modified, and imported into other projects for use. .. _ga4gh: http://genomicsandhealth.org/ .. _ga4gh_dwg: http://ga4gh.org/ .. _ga4gh_api: http://ga4gh.org/#/api

Index

I

L

N

O

P

R

S

T

V