MyVariant.info Documentation

Release v1

Su Lab

Aug 24, 2023

Contents

CHAPTER 1

Introduction

MyVariant.info provides simple-to-use REST web services to query/retrieve variant annotation data. It's designed with an emphasis on **simplicity** and **performance**.

CHAPTER 2

Quick start

MyVariant.info provides two simple web services: one for variant queries and the other for variant annotation retrieval. Both return results in JSON format.

2.1 Variant query service

2.1.1 URL

```
http://myvariant.info/v1/query
```

2.1.2 Examples

```
http://myvariant.info/v1/query?q=rs58991260
http://myvariant.info/v1/query?q=chr1:69000-70000
http://myvariant.info/v1/query?q=dbsnp.vartype:snp
http://myvariant.info/v1/query?q=_exists_:dbsnp  AND NOT dbsnp.vartype:indel
http://myvariant.info/v1/query?q=dbnsfp.polyphen2.hdiv.score:>0.99 AND chrom:1
http://myvariant.info/v1/query?q=cadd.gene.gene_id:ENSG00000113368&facets=cadd.
↪polyphen.cat&size=0
http://myvariant.info/v1/query?q=_exists_:dbsnp AND _exists_:cosmic
```

Hint: View nicely formatted JSON result in your browser with this handy add-on: JSON formatter for Chrome or JSONView for Firefox.

2.1.3 To learn more

- You can read the full description of our query syntax here.

- Try it live on interactive API page.
- Play with our demo application.
- Batch queries? Yes, you can. do it with a POST request.

2.2 Variant annotation service

2.2.1 URL

```
http://myvariant.info/v1/variant/<variant_id>
```

"<*variant_id*>" is an HGVS name based variant id using genomic location based on hg19 human genome assembly..

2.2.2 Examples

```
http://myvariant.info/v1/variant/chr1:g.35367G>A
http://myvariant.info/v1/variant/chr7:g.55241707G>T
http://myvariant.info/v1/variant/chr9:g.107620835G>A
http://myvariant.info/v1/variant/chr1:g.160145907G>T
http://myvariant.info/v1/variant/chr16:g.28883241A>G
http://myvariant.info/v1/variant/chr3:g.49721532G>A
```

2.2.3 To learn more

- You can read the full description of our query syntax here.
- Try it live on interactive API page.
- Play with our demo application.
- Yes, batch queries via POST request as well.

CHAPTER 3

Documentation

3.1 Variant annotation data

3.1.1 Data sources

We currently obtain variant annotation data from several data resources and keep them up-to-date, so that you don't have to do it:

Total hg19 variants loaded: **N/A**

Source	version	# of variants	key name*
dbNSFP	-	-	dbnsfp
dbSNP	-	-	dbsnp
ClinVar	-	-	clinvar
EVS	-	-	evs
CADD	-	-	cadd
MutDB	-	-	mutdb
GWAS Catalog	-	-	gwassnps
COSMIC	-	-	cosmic
DOCM	-	-	docm
SNPedia	-	-	snpedia
EMVClass	-	-	emv
Scripps Wellderly	-	-	wellderly
EXAC	-	-	exac
GRASP	-	-	grasp
UniProt	-	-	uniprot
CIViC	-	-	civic
Cancer Genome Interpreter	-	-	cgi
genome Aggregation Database	-	-	gnomad_genome
genome Aggregation Database	-	-	gnomad_exome
Geno2MP	-	-	geno2mp

Total hg38 variants loaded: **N/A**

Source	version	# of variants	key name*
dbNSFP	-	-	dbnsfp
dbSNP	-	-	dbsnp
ClinVar	-	-	clinvar
EVS	-	-	evs
UniProt	-	-	uniprot
genome Aggregation Database	-	-	gnomad_genome
genome Aggregation Database	-	-	gnomad_exome

* key name: this is the key for the specific annotation data in a variant object.

The most updated information can be accessed here for hg19 and here for hg38.

Note: Each data source may have its own usage restrictions (e.g. CADD data are free for non-commercial use only). Please refer to the data source pages above for their specific restrictions.

3.1.2 Variant object

Variant annotation data are both stored and returned as a variant object, which is essentially a collection of fields (attributes) and their values:


```
{
  "_id": "chr1:g.35367G>A",
  "_version": 2,
  "cadd": {
    "alt": "A",
    "annotype": "NonCodingTranscript",
    "chrom": 1,
    "gene": {
      "cds": {
        "cdna_pos": 476,
        "rel_cdna_pos": 0.4
      },
      "feature_id": "ENST00000417324",
      "gene_id": "ENSG00000237613",
    },
    "ref": "G",
    "type": "SNV"
  },
  "dbnsfp": {
    "aa": {
      "aapos_sift": "ENSP00000409362:P44L",
      "alt": "L",
      "codonpos": 2,
      "pos": 44,
      "ref": "P",
      "refcodon": "CCG"
    },
    "alt": "A",
    "ancestral_allele": "G",
    "chrom": "1",
```


(continues on next page)

(continued from previous page)


```
    "ensembl": {
      "geneid": "ENSG00000237613",
      "transcriptid": "ENST00000417324"
    },
    "genename": "FAM138A",
    "hg19": {
      "end": 35367,
      "start": 35367
    }
  }
}
```


The example above omits many of the available fields. For a full example, check out this example variant, or try the interactive API page.

3.1.3 _id field

Each individual variant object contains an "**_id**" field as the primary key. We utilize the recommended nomenclature from Human Genome Variation Society to define the "**_id**" field in MyVariant.info. Specifically, we use HGVS's genomic reference sequence notation based on the current reference genome assembly (e.g. hg19 for human). The followings are brief representations of major types of genetic variants. More examples could be found at HVGS recommendations for the description of DNA sequence variants page.

Note: The default reference genome assembly is always human hg19 in MyVariant.info, so we only use "chr??" to represent the reference genomic sequence in "**_id**" field. The valid chromosomes representations are **chr1**, **chr2**, . . . , **chr22**, **chrX**, **chrY** and **chrMT**. Do not use *chr23* for *chrX*, *chr24* for *chrY*, or *chrM* for *chrMT*.

- SNV example:

```
chr1:g.35366C>T
```

  The above _id represents a C to T SNV on chromosome 1, genomic position 35366.

- Insertion example:

```
chr2:g.17142_17143insA
```

  The above _id represents that an A is inserted between genomic position 17142 and 17143 on chromosome 2.

- Deletion example:

```
chrMT:g.8271_8279del
```

  The above _id represents that a nine nucleotides deletion between genomic position 8271 and 8279 on chromosome MT. Note that we don't include the deleted sequence in the _id field in this case.

- Deletion/Insertion example:

```
chrX:g.14112_14117delinsTG
```

  The above _id represents that six nucleotides between genomic position 14112 and 14117 are replaced by TG.

3.1.4 _score field

You will often see a "_score" field in the returned variant object, which is the internal score representing how well the query matches the returned variant object. It probably does not mean much in variant annotation service when only one variant object is returned. In variant query service, by default, the returned variant hits are sorted by the scores in descending order.

3.1.5 _version field

Sometime, you will see a "*_version*" field in the returned variant object, e.g. from the v1/variant endpoint. This field is basically for our internal information purpose, not very useful to the end users. You can just ignore it.

But for those who are curious, here is the explanation. The value of this "*_version*" field can be a small integer like 1, 2, 5 etc. The number indicates the version history of this particular variant object (i.e. how many times this object was updated). Because each variant object is updated independently and incrementally only when the updates to that particular variant are available, the "*_version*" values differ across variant objects. Of course, from time to time, when we need to make a full-data release (with some huge updates), every variant object will be re-created and their "*_version*" values will all be reset to 1.

Please also note that we don't keep any older versions of a variant object, the one returned from the API request is always the latest one we have. The "*_version*" field just indicates how many times it was updated in the past (since our last full data release).

3.1.6 Available fields

The table below lists all of the possible fields that could be in a variant object, as well as all of their parents (for nested fields). If the field is indexed, it may also be directly queried, e.g.

```
q=dbnsfp.polyphen2.hdiv.score:>0.99
```

All fields can be used with _exists_ or _missing_ filters, e.g.

```
q=_exists_:dbsnp AND _exists_:cosmic
q=_missing_:wellderly
```

or as inputs to the fields parameter, e.g.

```
q=_exists_:dbsnp&fields=dbsnp.rsid,dbsnp.vartype
```

3.2 Data release notes

This page contains metadata about each MyVariant.info data release. Click a link to see more.

3.2.1 MyVariant Releases

3.3 Variant query service

This page describes the reference for MyVariant.info variant query web service. It's also recommended to try it live on our interactive API page.

3.3.1 Service endpoint

```
http://myvariant.info/v1/query
```

3.3.2 GET request

Query parameters

q

Required, passing user query. The detailed query syntax for parameter "**q**" we explained *below*.

fields

Optional, a comma-separated string to limit the fields returned from the matching variant hits. The supported field names can be found from any variant object (e.g. here). Note that it supports dot notation, and wildcards as well, e.g., you can pass "dbnsfp", "dbnsfp.genename", or "dbnsfp.aa.*". If "fields=all", all available fields will be returned. Default: "all".

size

Optional, the maximum number of matching variant hits to return (with a cap of 1000 at the moment). Default: 10.

from

Optional, the number of matching variant hits to skip, starting from 0. Default: 0

Hint: The combination of "**size**" and "**from**" parameters can be used to get paging for large query:

```
q=cdk*&size=50                    first 50 hits
q=cdk*&size=50&from=50            the next 50 hits
```

fetch_all

Optional, a boolean, which when TRUE, allows fast retrieval of all unsorted query hits. The return object contains a **_scroll_id** field, which when passed as a parameter to the query endpoint, returns the next 1000 query results. Setting **fetch_all** = TRUE causes the results to be inherently unsorted, therefore the **sort** parameter is ignored. For more information see *examples using fetch_all here*. Default: FALSE.

scroll_id

Optional, a string containing the **_scroll_id** returned from a query request with **fetch_all** = TRUE. Supplying a valid **scroll_id** will return the next 1000 unordered results. If the next results are not obtained within 1 minute of the previous set of results, the **scroll_id** becomes stale, and a new one must be obtained

with another query request with **fetch_all** = TRUE. All other parameters are ignored when the **scroll_id** parameter is supplied. For more information see *examples using scroll_id here*.

sort

Optional, the comma-separated fields to sort on. Prefix with "-" for descending order, otherwise in ascending order. Default: sort by matching scores in descending order.

facets

Optional, a single field or comma-separated fields to return facets, for example, "facets=cadd", "facets=cadd,dbsnp.vartype". See *examples of faceted queries here*.

facet_size

Optional, an integer (1 <= **facet_size** <= 1000) that specifies how many buckets to return in a faceted query.

callback

Optional, you can pass a "**callback**" parameter to make a JSONP call.

dotfield

Optional, can be used to control the format of the returned variant object. If "dotfield" is true, the returned data object is returned flattened (no nested objects) using dotfield notation for key names. Default: false.

email

Optional, if you are regular users of our services, we encourage you to provide us an email, so that we can better track the usage or follow up with you.

Query syntax

Examples of query parameter "**q**":

Simple queries

search for everything:

```
q=rs58991260                    # search for rsid
```

Fielded queries

```
q=chr1:69000-70000                          # for a genomic range
q=dbsnp.vartype:snp                         # for matching value on a specific field

q=dbnsfp.polyphen2.hdiv.pred:(D P)          # multiple values for a field
q=dbnsfp.polyphen2.hdiv.pred:(D OR P)       # multiple values for a field using OR

q=_exists_:dbsnp                            # having dbsnp field
q=_missing_:exac                            # missing exac field
```

Hint: For a list of available fields, see *here*.

Range queries

```
q=dbnsfp.polyphen2.hdiv.score:>0.99
q=dbnsfp.polyphen2.hdiv.score:>=0.99
q=exac.af:<0.00001
q=exac.af:<=0.00001

q=exac.ac.ac_adj:[76640 TO 80000]         # bounded (including 76640 and 80000)
q=exac.ac.ac_adj:{76640 TO 80000}         # unbounded
```

Wildcard queries

Wildcard character "*" or "?" is supported in either simple queries or fielded queries:

```
q=dbnsfp.genename:CDK?
q=dbnsfp.genename:CDK*
```

Note: Wildcard character can not be the first character. It will be ignored.

Scrolling queries

If you want to return ALL results of a very large query, sometimes the paging method described *above* can take too long. In these cases, you can use a scrolling query. This is a two-step process that turns off database sorting to allow very fast retrieval of all query results. To begin a scrolling query, you first call the query endpoint as you normally would, but with an extra parameter **fetch_all** = TRUE. For example, a GET request to:

```
http://myvariant.info/v1/query?q=cadd.phred:>50&fetch_all=TRUE
```

Returns the following object:

```
{
  "_scroll_id":
↪"c2NhbjsxMDs5MjQ2OTc2Ok5nM0d0czYzU1cyU0dUU1dFemo5Mmc7MTE1NTgyNjA6RV9La1c5WklSQy16cVFuRXFzcEV3dzs5M
↪",
```

(continues on next page)

(continued from previous page)

```
  "hits": [
    .
    .
    .
  ],
  "max_score": 0.0,
  "took": 84,
  "total": 58759
}
```

At this point, the first 1000 hits have been returned (of ~58,000 total), and a scroll has been set up for your query. To get the next batch of 1000 unordered results, simply execute a GET request to the following address, supplying the _scroll_id from the first step into the **scroll_id** parameter in the second step:

```
http://myvariant.info/v1/query?scroll_
↪id=c2NhbjsxMDsxMTU1NjY5MTpxSnFkTFdVQlJ6T1dRVzNQaWRzQkhROzExNTU4MjYxOkVfS2tXOVpJUkMtenFRbkVxc3BFd3c
```

Hint: Your scroll will remain active for 1 minute from the last time you requested results from it. If your scroll expires before you get the last batch of results, you must re-request the scroll_id by setting **fetch_all** = TRUE as in step 1.

Boolean operators and grouping

You can use **AND/OR/NOT** boolean operators and grouping to form complicated queries:

```
q=dbnsfp.polyphen2.hdiv.score:>0.99 AND chrom:1                    AND operator
q=_exists_:dbsnp AND NOT dbsnp.vartype:indel                       NOT operator
q=_exists_:dbsnp AND (NOT dbsnp.vartype:indel)                     grouping with␣
↪()
```

Genomic interval queries can be mixed in as well, but only when surrounded by **AND** operators, and cannot be used inside parentheses.:

```
q=dbnsfp.genename:CDK* AND chr2:39406300-39406400
q=chr2:39406300-39406400 AND dbnsfp.genename:CDK*
```

Escaping reserved characters

If you need to use these reserved characters in your query, make sure to escape them using a back slash (""):

```
+ - = && || > < ! ( ) { } [ ] ^ " ~ * ? : \ /
```

Returned object

A GET request like this:

```
http://myvariant.info/v1/query?q=chr1:69500-70000&fields=cadd.gene
```

should return hits as:

```
{
  "hits": [
    {
      "_id": "chr1:g.69511A>G",
      "_score": 7.2999496,
      "cadd": {
        "gene": {
          "ccds_id": "CCDS30547.1",
          "cds": {
            "cdna_pos": 421,
            "cds_pos": 421,
            "rel_cdna_pos": 0.46,
            "rel_cds_pos": 0.46
          },
          "feature_id": "ENST00000335137",
          "gene_id": "ENSG00000186092",
          "genename": "OR4F5",
          "prot": {
            "domain": "tmhmm",
            "protpos": 141,
            "rel_prot_pos": 0.46
          }
        }
      }
    },
    {
      "_id": "chr1:g.69538G>A",
      "_score": 0.78757036,
      "cadd": {
        "gene": {
          "ccds_id": "CCDS30547.1",
          "cds": {
            "cdna_pos": 448,
            "cds_pos": 448,
            "rel_cdna_pos": 0.49,
            "rel_cds_pos": 0.49
          },
          "feature_id": "ENST00000335137",
          "gene_id": "ENSG00000186092",
          "genename": "OR4F5",
          "prot": {
            "domain": "ndomain",
            "protpos": 150,
            "rel_prot_pos": 0.49
          }
        }
      }
    }
  ],
  "max_score": 7.2999496,
  "took": 2325,
  "total": 2
}
```


"**total**" in the output gives the total number of matching hits, while the actual hits are returned under "**hits**" field. "**size**" parameter controls how many hits will be returned in one request (default is 10). Adjust "**size**" parameter and "**from**" parameter to retrieve the additional hits.

Faceted queries

If you need to perform a faceted query, you can pass an optional "*facets*" parameter. For example, if you want to get the facets on species, you can pass "facets=taxid":

A GET request like this:

```
http://myvariant.info/v1/query?q=cadd.gene.gene_id:ENSG00000113368&facets=cadd.
↪polyphen.cat&size=0
```

should return hits as:


```
{
  "facets": {
    "cadd.polyphen.cat": {
      "_type": "terms",
      "missing": 797,
      "other": 0,
      "terms": [
        {
          "count": 1902,
          "term": "benign"
        },
        {
          "count": 998,
          "term": "probably_damaging"
        },
        {
          "count": 762,
          "term": "possibly_damaging"
        }
      ],
      "total": 3662
    }
  },
  "hits": [],
  "max_score": 0.0,
  "took": 29,
  "total": 4459
}
```


3.3.3 Batch queries via POST

Although making simple GET requests above to our variant query service is sufficient for most use cases, there are times you might find it more efficient to make batch queries (e.g., retrieving variant annotation for multiple variants). Fortunately, you can also make batch queries via POST requests when you need:

```
URL: http://myvariant.info/v1/query
HTTP method:  POST
```

Query parameters

q

Required, multiple query terms seperated by comma (also support "+" or white space), but no wildcard, e.g., 'q=rs58991260,rs2500'

scopes

Optional, specify one or more fields (separated by comma) as the search "scopes", e.g., "scopes=dbsnp.rsid", "scopes=dbsnp.rsid,dbnsfp.genename". The available "fields" can be passed to "**scopes**" parameter are *listed here*. Default:

fields

Optional, a comma-separated string to limit the fields returned from the matching variant hits. The supported field names can be found from any variant object. Note that it supports dot notation, and wildcards as well, e.g., you can pass "dbnsfp", "dbnsfp.genename", or "dbnsfp.aa.*". If "fields=all", all available fields will be returned. Default: "all".

email

Optional, if you are regular users of our services, we encourage you to provide us an email, so that we can better track the usage or follow up with you.

Example code

Unlike GET requests, you can easily test them from browser, make a POST request is often done via a piece of code. Here is a sample python snippet:

```
import httplib2
h = httplib2.Http()
headers = {'content-type': 'application/x-www-form-urlencoded'}
params = 'q=rs58991260,rs2500&scopes=dbsnp.rsid'
res, con = h.request('http://myvariant.info/v1/query', 'POST', params,␣
↪headers=headers)
```

Returned object

Returned result (the value of "con" variable above) from above example code should look like this:

```
[
{'_id': 'chr1:g.218631822G>A',
  'dbsnp': {'allele_origin': 'unspecified',
   'alleles': [{'allele': 'G', 'freq': 0.9784},
    {'allele': 'A', 'freq': 0.02157}],
   'alt': 'A',
   'chrom': '1',
   'class': 'SNV',
   'dbsnp_build': 129,
   'flags': ['ASP', 'G5', 'G5A', 'GNO', 'KGPhase1', 'KGPhase3', 'SLO'],
   'gmaf': 0.02157,
```

(continues on next page)

(continued from previous page)

```
   'hg19': {'end': 218631823, 'start': 218631822},
   'ref': 'G',
   'rsid': 'rs58991260',
   'validated': True,
   'var_subtype': 'ts',
   'vartype': 'snp'},
  'query': 'rs58991260',
  'wellderly': {'alleles': [{'allele': 'A', 'freq': 0.0025},
    {'allele': 'G', 'freq': 0.9975}],
   'alt': 'A',
   'chrom': '1',
   'gene': 'TGFB2',
   'genotypes': [{'count': 1, 'freq': 0.005, 'genotype': 'G/A'},
    {'count': 199, 'freq': 0.995, 'genotype': 'G/G'}],
   'hg19': {'end': 218631822, 'start': 218631822},
   'pos': 218631822,
   'ref': 'G',
   'vartype': 'snp'}},
 {'_id': 'chr11:g.66397320A>G',
  'dbsnp': {'allele_origin': 'unspecified',
   'alleles': [{'allele': 'A'}, {'allele': 'G'}],
   'alt': 'G',
   'chrom': '11',
   'class': 'SNV',
   'dbsnp_build': 36,
   'flags': ['ASP', 'INT', 'RV', 'U3'],
   'hg19': {'end': 66397321, 'start': 66397320},
   'ref': 'A',
   'rsid': 'rs2500',
   'validated': False,
   'var_subtype': 'ts',
   'vartype': 'snp'},
  'query': 'rs2500'}
]
```

Tip: "query" field in returned object indicates the matching query term.

If a query term has no match, it will return with "**notfound**" field as "**true**":

```
[
  ...,
  {'query': '...',
   'notfound': true},
  ...
]
```

3.4 Variant annotation service

This page describes the reference for the MyVariant.info variant annotation web service. It's also recommended to try it live on our interactive API page.

3.4.1 Service endpoint

```
http://myvariant.info/v1/variant
```

3.4.2 GET request

Obtaining the variant annotation via our web service is as simple as calling this URL:

```
http://myvariant.info/v1/variant/<variantid>
```

variantid above is an HGVS name based variant id using genomic location based on hg19 human genome assembly.

By default, this will return the complete variant annotation object in JSON format. See *here* for an example and *here* for more details. If the input **variantid** is not valid, 404 (NOT FOUND) will be returned.

Optionally, you can pass a "**fields**" parameter to return only the annotation you want (by filtering returned object fields):

```
http://myvariant.info/v1/variant/chr1:g.35367G>A?fields=cadd
```

"**fields**" accepts any attributes (a.k.a fields) available from the variant object. Multiple attributes should be separated by commas. If an attribute is not available for a specific variant object, it will be ignored. Note that the attribute names are case-sensitive.

Just like the variant query service, you can also pass a "**callback**" parameter to make a JSONP call.

Query parameters

fields

Optional, can be a comma-separated fields to limit the fields returned from the variant object. If "fields=all", all available fields will be returned. Note that it supports dot notation as well, e.g., you can pass "cadd.gene". Default: "fields=all".

callback

Optional, you can pass a "**callback**" parameter to make a JSONP call.

filter

Alias for "fields" parameter.

email

Optional, if you are regular users of our services, we encourage you to provide us an email, so that we can better track the usage or follow up with you.

Returned object

A GET request like this:

```
http://myvariant.info/v1/variant/chr1:g.35367G>A
```

should return a variant object below:


```
{
  "_id": "chr1:g.35367G>A",
  "_version": 2,
  "cadd": {
    "alt": "A",
    "annotype": "NonCodingTranscript",
    "bstatistic": 994,
    "chmm": {
      "bivflnk": 0.0,
      "enh": 0.0,
      "enhbiv": 0.0,
      "het": 0.0,
      "quies": 1.0,
      "reprpc": 0.0,
      "reprpcwk": 0.0,
      "tssa": 0.0,
      "tssaflnk": 0.0,
      "tssbiv": 0.0,
      "tx": 0.0,
      "txflnk": 0.0,
      "txwk": 0.0,
      "znfrpts": 0.0
    },
    "chrom": 1,
    "consdetail": "non_coding_exon,nc",
    "consequence": "NONCODING_CHANGE",
    "consscore": 5,
    "cpg": 0.03,
    "dna": {
      "helt": -2.04,
      "mgw": 0.01,
      "prot": 1.54,
      "roll": -0.63
    },
    "encode": {
      "exp": 31.46,
      "h3k27ac": 23.44,
      "h3k4me1": 23.8,
      "h3k4me3": 8.6
    },
    "exon": "2/3",
    "fitcons": 0.577621,
    "gc": 0.48,
    "gene": {
      "cds": {
        "cdna_pos": 476,
        "rel_cdna_pos": 0.4
      },
      "feature_id": "ENST00000417324",
      "gene_id": "ENSG00000237613",
```


(continues on next page)

(continued from previous page)


```
      "genename": "FAM138A"
    },
    "gerp": {
      "n": 1.29,
      "s": -0.558
    },
    "isknownvariant": "FALSE",
    "istv": "FALSE",
    "length": 0,
    "mapability": {
      "20bp": 0,
      "35bp": 0
    },
    "min_dist_tse": 122,
    "min_dist_tss": 707,
    "mutindex": 70,
    "phast_cons": {
      "mammalian": 0.003,
      "primate": 0.013,
      "vertebrate": 0.003
    },
    "phred": 1.493,
    "phylop": {
      "mammalian": -0.155,
      "primate": 0.151,
      "vertebrate": -0.128
    },
    "pos": 35367,
    "rawscore": -0.160079,
    "ref": "G",
    "scoresegdup": 0.99,
    "segway": "D",
    "type": "SNV"
  },
  "dbnsfp": {
    "aa": {
      "aapos_sift": "ENSP00000409362:P44L",
      "alt": "L",
      "codonpos": 2,
      "pos": 44,
      "ref": "P",
      "refcodon": "CCG"
    },
    "alt": "A",
    "ancestral_allele": "G",
    "cadd": {
      "phred": 1.493,
      "raw": -0.160079,
      "raw_rankscore": 0.05963
    },
    "cds_strand": "-",
    "chrom": "1",
    "ensembl": {
      "geneid": "ENSG00000237613",
      "transcriptid": "ENST00000417324"
    },
    "fold-degenerate": 0,
```


(continues on next page)

(continued from previous page)


```
    "genename": "FAM138A",
    "gerp++": {
      "nr": 1.29,
      "rs": -0.558,
      "rs_rankscore": 0.11796
    },
    "hg18": {
      "end": 25230,
      "start": 25230
    },
    "hg19": {
      "end": 35367,
      "start": 35367
    },
    "hg38": {
      "end": 35367,
      "start": 35367
    },
    "lr": {
      "pred": "T",
      "rankscore": 0.32941,
      "score": 0.0846
    },
    "mutationtaster": {
      "converted_rankscore": 0.10124,
      "pred": "N",
      "score": 1.0
    },
    "phastcons": {
      "100way": {
        "vertebrate": 0.001,
        "vertebrate_rankscore": 0.1272
      },
      "46way": {
        "placental": 0.003,
        "placental_rankscore": 0.11579,
        "primate": 0.336,
        "primate_rankscore": 0.28762
      }
    },
    "phylop": {
      "100way": {
        "vertebrate": 0.055,
        "vertebrate_rankscore": 0.14229
      },
      "46way": {
        "placental": -0.143,
        "placental_rankscore": 0.11334,
        "primate": 0.175,
        "primate_rankscore": 0.17021
      }
    },
    "provean": {
      "converted_rankscore": 0.92415,
      "pred": "D",
      "score": -6.73
    },
```


(continues on next page)

(continued from previous page)


```
    "radialsvm": {
      "pred": "T",
      "rankscore": 0.32188,
      "score": -0.9916
    },
    "ref": "G",
    "reliability_index": 5,
    "sift": {
      "converted_rankscore": 0.2575,
      "pred": "T",
      "score": 0.13
    },
    "siphy_29way": {
      "logodds": 3.5364,
      "logodds_rankscore": 0.07795,
      "pi": {
        "a": 0.5429,
        "c": 0.0,
        "g": 0.4571,
        "t": 0.0
      }
    },
    "vest3": {
      "rankscore": 0.16309,
      "score": 0.137
    }
  }
}
```


3.4.3 Batch queries via POST

Although making simple GET requests above to our variant query service is sufficient in most use cases, there are some times you might find it's easier to batch query (e.g., retrieving variant annotations for multiple variants). Fortunately, you can also make batch queries via POST requests when you need:

```
URL: http://myvariant.info/v1/variant
HTTP method:  POST
```

Query parameters

ids

Required. Accept multiple HGVS variant ids separated by comma, e.g., "ids=chr1:g.35367C>T,chr7:g.55241707G>T,chr16:g.28883241A>G". Note that currently we only take the input ids up to **1000** maximum, the rest will be omitted.

fields

Optional, can be a comma-separated fields to limit the fields returned from the matching hits. If "fields=all", all available fields will be returned. Note that it supports dot notation as well, e.g., you can pass "dbnsfp", "dbnsfp.genename", or "dbnsfp.aa.*". Default: "all".

email

Optional, if you are regular users of our services, we encourage you to provide us an email, so that we can better track the usage or follow up with you.

Example code

Unlike GET requests, you can easily test them from browser, make a POST request is often done via a piece of code, still trivial of course. Here is a sample python snippet:

```
import httplib2
h = httplib2.Http()
headers = {'content-type': 'application/x-www-form-urlencoded'}
params = 'ids=chr16:g.28883241A>G,chr1:g.35367G>A&fields=dbnsfp.genename,cadd.gene'
res, con = h.request('http://myvariant.info/v1/variant', 'POST', params,␣
↪headers=headers)
```

Returned object

Returned result (the value of "con" variable above) from above example code should look like this:

```
[
  {
    "_id": "chr16:g.28883241A>G",
    "cadd": {
      "gene": {
        "ccds_id": "CCDS53996.1",
        "cds": {
          "cdna_pos": 1889,
          "cds_pos": 1450,
          "rel_cdna_pos": 0.61,
          "rel_cds_pos": 0.64
        },
        "feature_id": "ENST00000322610",
        "gene_id": "ENSG00000178188",
        "genename": "SH2B1",
        "prot": {
          "protpos": 484, "rel_prot_pos": 0.64
        }
      }
    },
    "dbnsfp": {
      "genename": "SH2B1"
    },
    "query": "chr16:g.28883241A>G"
  },
  {
    "_id": "chr1:g.35367G>A",
    "cadd": {
      "gene": {
        "cds": {
          "cdna_pos": 476,
          "rel_cdna_pos": 0.4
        },
        "feature_id": "ENST00000417324",
```

(continues on next page)

(continued from previous page)


```
        "gene_id": "ENSG00000237613",
        "genename": "FAM138A"
      }
    },
    "dbnsfp": {
      "genename": "FAM138A"
    },
    "query": "chr1:g.35367G>A"
  }
]
```


3.5 Server response

The MyVariant.info server returns a variety of query responses, and response status codes. They are listed here.

Note: These examples show query responses using the python requests package.

3.5.1 Status code *200*

A **200** status code indicates a successful query, and is accompanied by the query response payload.

```
In [1]: import requests

In [2]: r = requests.get('http://myvariant.info/v1/query?q=_exists_:clinvar')

In [3]: r.status_code
Out[3]: 200

In [4]: data = r.json()

In [5]: data.keys()
Out[5]: dict_keys(['total', 'max_score', 'took', 'hits'])
```

3.5.2 Status code *400*

A **400** status code indicates an improperly formed query, and is accompanied by a response payload describing the source of the error.

```
In [6]: r = requests.get('http://myvariant.info/v1/query?q=_exists_:clinvar&size=u')

In [7]: r.status_code
Out[7]: 400

In [8]: data = r.json()

In [9]: data
Out[9]:
{'error': "Expected 'size' parameter to have integer type.  Couldn't convert 'u' to␣
↪integer",
 'success': False}
```

3.5.3 Status code *404*

A **404** status code indicates either an unrecognized URL, as in (*/query* is misspelled */quer* resulting in an unrecognized URL):

```
In [10]: r = requests.get('http://myvariant.info/v1/quer?q=_exists_:clinvar')

In [11]: r.status_code
Out[11]: 404
```

or, for the **/variant** endpoint, a **404** status code could be from querying for a nonexistent HGVS ID, as in:

```
In [12]: r = requests.get('http://myvariant.info/v1/variant/5')

In [13]: r.status_code
Out[13]: 404

In [14]: data = r.json()

In [15]: data
Out[15]:
{'error': "ID '5' not found",
 'success': False}
```

3.5.4 Status code *5xx*

Any **5xx** status codes are the result of uncaught query errors. Ideally, these should never occur. We routinely check our logs for these types of errors and add code to catch them, but if you see any status **5xx** responses, please submit a bug report to help@myvariant.info.

3.6 Third-party packages

This page lists the third-party packages/modules built upon the MyVariant.info services.

3.6.1 MyVariant python module

"myvariant" is an easy-to-use Python wrapper to access MyVariant.info services.

You can install it easily using either pip or easy_install:

```
pip install myvariant               #this is preferred
```

or:

```
easy_install myvariant
```

This is a brief example:

```
In [1]: import myvariant

In [2]: mv = myvariant.MyVariantInfo()

```

(continues on next page)

(continued from previous page)

```
In [3]: mv.getvariant('chr1:g.35367G>A')
Out[3]:
{'_id': 'chr1:g.35367G>A',
 '_version': 1,
 'cadd': {'alt': 'A',
  'annotype': 'NonCodingTranscript',
  'bstatistic': 994,
  'chmm': {'bivflnk': 0.0,
   'enh': 0.0,
   'enhbiv': 0.0,
   'het': 0.0,
   'quies': 1.0,
   'reprpc': 0.0,
   'reprpcwk': 0.0,
   'tssa': 0.0,
   'tssaflnk': 0.0,
   'tssbiv': 0.0,
   'tx': 0.0,
   'txflnk': 0.0,
   'txwk': 0.0,
   'znfrpts': 0.0},
  'chrom': 1,
  'consdetail': 'non_coding_exon,nc',
  'consequence': 'NONCODING_CHANGE',
  'consscore': 5,
  'cpg': 0.03,
  'dna': {'helt': -2.04, 'mgw': 0.01, 'prot': 1.54, 'roll': -0.63},
  'encode': {'exp': 31.46, 'h3k27ac': 23.44, 'h3k4me1': 23.8, 'h3k4me3': 8.6},
  'exon': '2/3',
  'fitcons': 0.577621,
  'gc': 0.48,
  'gene': {'cds': {'cdna_pos': 476, 'rel_cdna_pos': 0.4},
   'feature_id': 'ENST00000417324',
   'gene_id': 'ENSG00000237613',
   'genename': 'FAM138A'},
  'gerp': {'n': 1.29, 's': -0.558},
  'isknownvariant': 'FALSE',
  'istv': 'FALSE',
  'length': 0,
  'mapability': {'20bp': 0, '35bp': 0},
  'min_dist_tse': 122,
  'min_dist_tss': 707,
  'mutindex': 70,
  'phast_cons': {'mammalian': 0.003, 'primate': 0.013, 'vertebrate': 0.003},
  'phred': 1.493,
  'phylop': {'mammalian': -0.155, 'primate': 0.151, 'vertebrate': -0.128},
  'pos': 35367,
  'rawscore': -0.160079,
  'ref': 'G',
  'scoresegdup': 0.99,
  'segway': 'D',
  'type': 'SNV'},
 'snpeff': {'ann': [{'effect': 'non_coding_exon_variant',
    'feature_id': 'NR_026818.1',
    'feature_type': 'transcript',
    'gene_id': 'FAM138A',
    'gene_name': 'FAM138A',
```

(continues on next page)

(continued from previous page)

```
    'hgvs_c': 'n.476C>T',
    'putative_impact': 'MODIFIER',
    'rank': '2',
    'total': '3',
    'transcript_biotype': 'Noncoding'},
   {'effect': 'non_coding_exon_variant',
    'feature_id': 'NR_026820.1.2',
    'feature_type': 'transcript',
    'gene_id': 'FAM138F.2',
    'gene_name': 'FAM138F',
    'hgvs_c': 'n.476C>T',
    'putative_impact': 'MODIFIER',
    'rank': '2',
    'total': '3',
    'transcript_biotype': 'Noncoding'}]},
 'vcf': {'alt': 'A', 'position': '35367', 'ref': 'G'}}
```

See https://pypi.python.org/pypi/myvariant for more details.

3.6.2 MyVariant R package

An R wrapper for the MyVariant.info API is available in Bioconductor since v3.2. To install:

```
source("https://bioconductor.org/biocLite.R")
biocLite("myvariant")
```

To view documentation for your installation, enter R and type:

```
browseVignettes("myvariant")
```

For more information, visit the Bioconductor myvariant page.

3.6.3 MyVariant Node.js package

myvariantjs is a Node.js wrapper for the MyVariant.info API, developed and maintained by Larry Hengl. To install:

```
npm install myvariantjs --save
```

Some brief usage examples:

```
var mv = require('myvariantjs');
mv.getvariant('chr9:g.107620835G>A');
mv.getvariant('chr9:g.107620835G>A', ['dbnsfp.genename', 'cadd.phred']);

mv.getvariants("chr1:g.866422C>T,chr1:g.876664G>A,chr1:g.69635G>C");  // string of␣
↪delimited ids
mv.getvariants(["chr1:g.866422C>T", "chr1:g.876664G>A","chr1:g.69635G>C"]);

mv.query("chr1:69000-70000", {fields:'dbnsfp.genename'});
mv.query("dbsnp.rsid:rs58991260", {fields:'dbnsfp'});

mv.querymany(['rs58991260', 'rs2500'], 'dbsnp.rsid');
mv.querymany(['RCV000083620', 'RCV000083611', 'RCV000083584'], 'clinvar.rcv_accession
↪');
```

For more information, visit its API and usage docs, and its github code repository.

You can also check out this neat demo application developed by Larry using this myvariantjs package.

3.6.4 Another MyVariant.info python module

This is another python wrapper of MyVariant.info services created by Brian Schrader. The repository is available here.

You can install this package with PyPI like this:

```
pip install myvariant-api
```

3.6.5 A JBrowse plugin for MyVariant.info and MyGene.info

JBrowse provides a fast, embeddable genome browser built completely with JavaScript and HTML5.

Colin from the JBrowse team made a very nice plugin to visualize the gene and variant annotations in JBrowse Genome Browser, using the data served from both MyGene.info and MyVariant.info APIs.

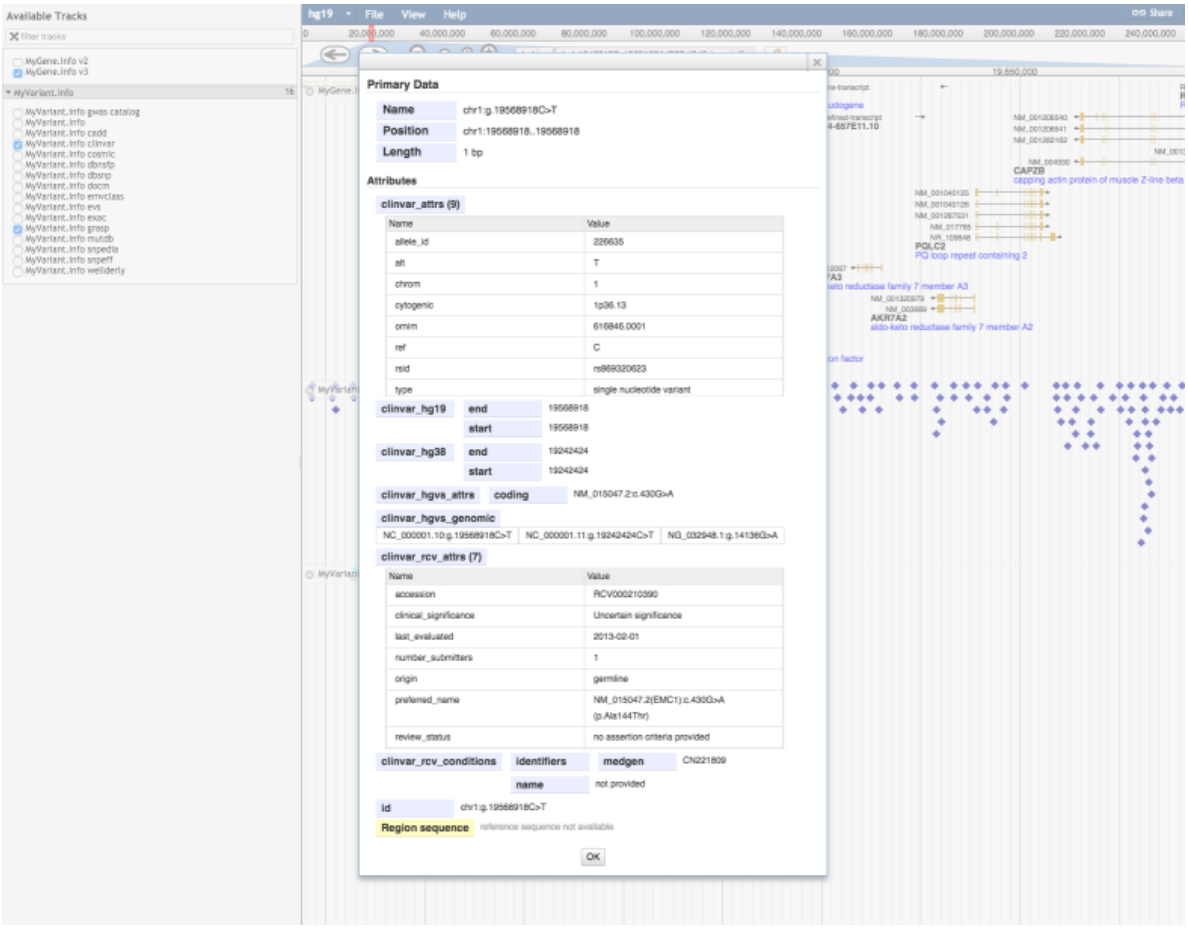
- Live demo

  To see it live, here is the demo site. It has been tested with hg38, hg19, and zebrafish and has mygene.info and myvariant.info integrations

- Source code

  https://github.com/elsiklab/myvariantviewer

- A screenshot


Available Tracks
hg19
File
View
Help
Share
Primary Data
Name
chr1:g.19568918C>T
Position
chr1:19568918..19568918
Length
1 bp
Attributes
clinvar_attrs (9)
Name
Value
allele_id
226635
alt
T
chrom
1
cytogenic
1p36.13
omim
616846.0001
ref
C
rsid
rs869320623
type
single nucleotide variant
clinvar_hg19
end
19568918
start
19568918
clinvar_hg38
end
19242424
start
19242424
clinvar_hgvs_attrs
coding
NM_015047.2:c.430G>A
clinvar_hgvs_genomic
NC_000001.10:g.19568918C>T
NC_000001.11:g.19242424C>T
NG_032948.1:g.14136G>A
clinvar_rcv_attrs (7)
Name
Value
accession
RCV000210390
clinical_significance
Uncertain significance
last_evaluated
2013-02-01
number_submitters
1
origin
germline
preferred_name
NM_015047.2(EMC1):c.430G>A (p.Ala144Thr)
review_status
no assertion criteria provided
clinvar_rcv_conditions
identifiers
medgen
CN221809
name
not provided
id
chr1:g.19568918C>T
Region sequence
reference sequence not available
OK
CAPZB
capping actin protein of muscle Z-line beta
POLC2
PO loop repeat containing 2
AKR7A2
aldo-keto reductase family 7 member A2

CHAPTER 4

Related links

- github repository