Allele Registry

API specification

version 0.08.04

Table of Contents

Introduction

Allele Registry provides URIs for canonical alleles defined at the level of nucleic acid sequences (genomic and transcript alleles) or at the protein level (amino acid sequences). Different labels and definitions of the same allele are always represented by the same URI. Canonical allele embraces various names of the same allele and its definitions in the context of different reference sequences (both assemblies and transcripts). Nucleic acid and amino acid canonical alleles are defined in separate spaces and never share the same URI.
This document describes API for Allele Registry that allows querying as well as registering alleles and obtaining their URI in real time. The API is based on HTTP protocol and always returns data in JSON format. New fields may be added in the future, so developers using this API should assume that all structures may contain additional fields not described in this document.
Allele Registry is identified on the Internet by DNS name. This address will be denoted in this document by `{ServerName}`. Described API is currently available under the address http://reg.genome.network and http://reg.clinicalgenome.org. There is also test server available at the location http://reg.test.genome.network.

Examples

This documentation is enriched with examples. All of them may be run with included code snippets. Provided examples are coded in different programming languages summarized in the following table:

ruby	All included ruby code snippets should work with ruby version >= 1.8.7. The following library must be loaded to execute ruby examples:
	
	`require 'net/http'`
	`require 'digest/sha1'`

python	Python code snippets should work with python version >= 2.7. The following library are needed to run python code snippets:
	
	`import requests`
	`import hashlib`
	`import time`
	
	The library “request” is not a part of the Python Standard Library and probably must be installed separately (it should be available through Linux package manager).

bash	These sections contain sequence of commands which may be run from bash console (by copy & paste). They require some additional tools like curl or sha1sum and depend on standard tools like echo, cut etc.. Different behavior of these dependencies may perturb some examples. However provided code snippets should work on the majority of modern Linux distributions.

All HTTP GET requests included in examples can be also run in any Internet browser.

Authentication

While all HTTP GET and HTTP POST requests are accepted without authentication, active account in Allele Registry is required for sending all HTTP PUT requests. Three special parameters must be added to every request that needs an authentication:

1. gbLogin – user login
2. gbTime – current time saved as integer number of seconds since the Epoch
3. gbToken – special token calculated from original request URL, gbLogin, gbTime and user password

The parameter gbToken is calculated in the following way:

```
SHA1_hex(url + SHA1_hex(gbLogin + password) + gbTime)
```

where url is the original request (without gbLogin, gbTime and gbToken, if there is no parameters it must have question mark at the end), operator + denotes simple string concatenation and SHA1_hex(...) denotes hexadecimal representation of SHA1 calculated on given ASCII string.
In the table below there are sample code snippet which may be used for preparing a request with authentication (variables url, login and password must be set in advance).

ruby	`identity = Digest::SHA1.hexdigest("#{login}#{password}")`
	`gbTime = Time.now.to_i.to_s`
	`token = Digest::SHA1.hexdigest("#{url}#{identity}#{gbTime}")`
	`request = "#{url}&gbLogin=#{login}&gbTime=#{gbTime}&gbToken=#{token}"`

python	`identity = hashlib.sha1((login + password).encode('utf-8')).hexdigest()`
	`gbTime = str(int(time.time()))`
	`token = hashlib.sha1((url+identity+gbTime).encode('utf-8')).hexdigest()`
	`request = url+'&gbLogin='+login+'&gbTime='+gbTime+'&gbToken='+token`

bash	`IDENTITY=`echo -n "${LOGIN}${PASSWORD}" | sha1sum | cut -d \ -f 1` `TIME=`date +%s | tr -d "\n"` `TOKEN=`echo -n "${URL}${IDENTITY}${TIME}" \| sha1sum \| cut -d \ -f 1` `REQUEST="${URL}&gbLogin=${LOGIN}&gbTime=${TIME}&gbToken=${TOKEN}"`

Error responses

All responses with status different than HTTP SUCCESS contain a body with a single JSON object consisting of the following fields:

Name	Type	When returned?	Description
HttpStatusCode (obsolete)	integer	always	HTTP status code
HttpStatusName (obsolete)	string	always	HTTP status name
errorType	string	always	Error type, see the table below
description	string	always	Description of error type given above
message	string	may be missing	Detailed information about error

Returned error object may also contain some additional fields, depending on the errorType. The field

errorType always contains one of the short strings from the table below.

errorType	description	HTTP status
NotFound	The system does not contain any data about requested resource.	404 (Not Found)
AuthorizationError	Access denied because of authorization failure.	403 (Forbidden)
HgvsParsingError	Given HGVS expressions cannot be parsed. It is incorrect or not supported.	400 (Bad Request)
IncorrectHgvsPosition	Position given in HGVS expression is incorrect.	400 (Bad Request)
IncorrectReferenceAllele	Given allele from reference sequence is incorrect. It does not match actual sequence at given position.	400 (Bad Request)
NoConsistentAlignment	Given allele cannot be mapped in consistent way to reference genome.	400 (Bad Request)
UnknownCDS	The boundary of coding sequence for given transcript is not known.	400 (Bad Request)
UnknownGene	Given reference sequence is not assigned to any gene.	400 (Bad Request)
UnknownReferenceSequence	Given reference sequence is not known.	400 (Bad Request)
VcfParsingError	Sent VCF file cannot be parsed. It is incorrect or contains unsupported features.	400 (Bad Request)
InternalServerError	Internal error occurred. Please, report it as an error.	500 (Internal Server Error)

Objects exposed by this API

There are three main types of objects accessible through API and uniquely identified by the following URIs:

- canonical alleles – `http://{ServerName}/allele/{id}`
- genes – `http://{ServerName}/gene/{id}`
- reference sequences – `http://{ServerName}/refseq/{id}`

HTTP request to object's URI returns HTTP NOT FOUND status if there is no object with a given URI or HTTP OK status with JSON representation of the object in response's body. The formats of possible responses are described in the following sections. All objects may contain some additional fields, not described in the documentation below. No assumption should be made about these fields.

Canonical Allele

URL: `http://{ServerName}/allele/{id}`
The successful response contains exactly one object with the following fields.

Name	Type	When returned?	Description
@id	An allele URI	always	The URI of the allele.
type	“nucleotide” or “amino-acid”	always	The type of the allele.
activeUris	An array of allele URIs	if and only if the URI is inactive	The list of active allele URIs that superseded the current one.
externalRecords	Object externalRecords (see below)	only if the URI is active and there are known links to similar records in other systems	Known records from other systems with the allele.
genomicAlleles	A non-empty array of objects alleleDefinition (see below)	only if the URI is active and alleleType is set to “nucleotide”, omitted when empty	A list of known definitions of the allele in the context of genomic reference sequences.
transcriptAlleles	A non-empty array of objects alleleDefinition (see below)	only if the URI is active and the alleleType is set to “nucleotide”, omitted when empty	A list of known definitions of the allele in the context of transcript reference sequences.
aminoAcidAlleles	A non-empty array of objects alleleDefinition (see below)	if and only if the URI is active and the alleleType is set to “amino-acid”	A list of known definitions of the allele in the context of amino-acid reference sequences.

externalRecords – object used in definition of canonical allele object above:

Name	Type	When returned?	Description
dbSNP	An object	Only if non-empty	May contains any subset of the following fields: • @id – link to record in dbSNP • rs – rs number from dbSNP
ClinVar	An object	Only if non-empty	May contains any subset of the following fields: • @id – link to Variation record in ClinVar • variationId • alleleId • preferredName • RCV

alleleDefinition – object used in definition of canonical allele object above:

Name	Type	When returned?	Description
hgvs	An array of strings	always	Non-empty list of HGVS expressions defining the allele in the context of single reference sequence.
referenceSequence	A refseq URI	always	The URI of the reference sequence.
coordinates	A non-empty array of objects coordinates (see below)	always	A list of subsequences of reference sequence belonging to the allele

coordinates – object used in definition of allele object above:

Name	Type	When returned?	Description
start	A non-negative integer	always	Begin of a reference subsequence covered by allele.
end	A non-negative integer	always	End of a reference subsequence covered by allele.
startIntronOffset	A non-negative integer	if and only if the reference sequence is transcript and the allele begins inside an intron	Distance (offset) of start position in an intron to the nearest exon.
startIntronDirection	“+” or “-”	if and only if the reference sequence is transcript and the allele begins inside an intron	Direction of the offset defined above.
endIntronOffset	A non-negative integer	if and only if the reference sequence is transcript and the allele ends inside an intron	Distance (offset) of end position in an intron to the nearest exon.
endIntronDirection	“+” or “-”	if and only if the reference sequence is transcript and the allele ends inside an intron	Direction of the offset defined above.
referenceAllele	String consisting of letters: 'A', 'C', 'G', 'T' for genomic and transcript alleles or sequence of protein symbols for amino-acid alleles	always	Original reference subsequence defined by the coordinates above.
allele	String consisting of letters: 'A', 'C', 'G', 'T' for genomic and transcript alleles or sequence of protein symbols for amino-acid alleles	always	Sequence put in place of reference subsequence defined above.

Example 1:

request: HTTP GET http://reg.test.genome.network/allele/CA012345

ruby	`url = 'http://reg.test.genome.network/allele/CA012345'`
	`res = Net::HTTP.get_response(URI(url))`
	`print res.body`

python	`res = requests.get('http://reg.test.genome.network/allele/CA012345')` `print(res.text)`
bash	`curl -X GET "http://reg.test.genome.network/allele/CA012345"`

response:


```
{
  "@context": "http://reg.test.genome.network/schema/allele.jsonld",
  "@id": "http://reg.test.genome.network/allele/CA012345",
  "type": "nucleotide",
  "externalRecords": {
    "dbSNP": {
      "@id": "http://www.ncbi.nlm.nih.gov/snp/749469486",
      "rs": "749469486"
    },
    "ClinVar": {
      "@id": "http://www.ncbi.nlm.nih.gov/clinvar/variation/186550",
      "variationId": "186550",
      "alleleId": "183678",
      "preferredHgvs": "NM_000059.3(BRCA2):c.1543A>G (p.Thr515Ala)",
      "RCV": "RCV000166164"
    }
  },
  "genomicAlleles": [
    {
      "hgvs": [ "NC_000013.11:g.32333021A>G" ],
      "referenceSequence": "http://reg.test.genome.network/refseq/RS542947913077",
      "coordinates": [
        {
          "end": 32333021,
          "allele": "G",
          "start": 32333020,
          "referenceAllele": "A"
        }
      ]
    }
  ],
  "transcriptAlleles": [
    {
      "coordinates": [
        {
          "end": 1770,
          "allele": "G",
          "start": 1769,
          "referenceAllele": "A"
        }
      ],
      "referenceSequence": "http://reg.test.genome.network/refseq/RS938330737581",
      "hgvs": [ "NM_000059.3:c.1543A>G" ]
    }
  ]
}
```

Reference Sequence

`URL: http://{ServerName}/refseq/{id}`
Fields description:

Name	Type	When returned?	Description
@id	A refseq URI	always	The URI of the reference sequence.
externalRecords	Object externalRecords	only if there are known links to similar records in other systems	Known records from other systems with the reference sequence.
type	“chromosome”or “transcript” or “amino-acid”	always	
referenceGenome	“NCBI36” or “GRCh37” or “GRCh38”	if and only if the field “type” is set to “chromosome”	The genome build in which the chromosomal reference sequence is referenced.
chromosome	one of the strings: “1”, “2”, …, “22”, “X”, “Y”, “MT”	if and only if the the field “type” is set to “chromosome”	
gene	A gene URI	only if the type is “transcript”, may be omitted	The URI of a gene associated with this transcript reference sequence.

Example 1:

request: HTTP GET http://reg.test.genome.network/refseq/RS000065

ruby	`url = 'http://reg.test.genome.network/refseq/RS000065'`
	`res = Net::HTTP.get_response(URI(url))`
	`print res.body`

python	`url = 'http://reg.test.genome.network/refseq/RS000065`
	`res = requests.get(url)`
	`print(res.text)`

bash	`curl -X GET "http://reg.test.genome.network/refseq/RS000065"`

response:


```
{
  "@context": "http://reg.test.genome.network/schema/refseq.jsonld",
  "@id": "http://reg.test.genome.network/refseq/RS000065",
  "type": "chromosome",
  "externalRecords": {
    "NCBI": {
      "@id": "http://www.ncbi.nlm.nih.gov/nuccore/NC_000017.11",
      "id": "NC_000017.11"
    }
  },
  "referenceGenome": "GRCh38",
  "chromosome": "17"
}
```

Example 2:

request: HTTP GET http://reg.test.genome.network/refseq/RS011494

response:


```
{
  "@context": "http://reg.test.genome.network/schema/refseq.jsonld",
  "@id": "http://reg.test.genome.network/refseq/RS011494",
  "type": "transcript",
  "externalRecords": {
    "LRG": {
      "@id":
"http://ftp.ebi.ac.uk/pub/databases/lrgex/LRG_321.xml#transcripts_anchor",
      "id": "LRG_321t6"
    },
    "NCBI": {
      "@id": "http://www.ncbi.nlm.nih.gov/nuccore/NM_001126116.1",
      "id": "NM_001126116.1"
    }
  },
  "gene": "http://reg.test.genome.network/gene/GN11998"
}
```


Example 3:

request: HTTP GET http://reg.test.genome.network/refseq/RS167707

response:


```
{
  "@context": "http://reg.test.genome.network/schema/refseq.jsonld",
  "@id": "http://reg.test.genome.network/refseq/RS167707",
  "type": "amino-acid",
  "externalRecords": {
    "NCBI": {
      "@id": "www.ncbi.nlm.nih.gov/nuccore/NP_001813.1",
      "id": "NP_001813.1"
    }
  }
}
```

Gene

URL: `http://{ServerName}/gene/{id}`
Fields description:

Name	Type	When returned?	Description
@id	A gene URI	always	The URI of the gene.
externalRecords	Object externalRecords	only if there are known links to similar records in other systems	Known records from other systems with the gene.
names	An array of strings	if not empty	A list of known gene's names (labels) not mentioned in the externalRecords.

Example:

request: HTTP GET http://reg.test.genome.network/gene/GN11998

ruby	`url = 'http://reg.test.genome.network/gene/GN11998'`
	`res = Net::HTTP.get_response(URI(url))`
	`print res.body`

python	`res = requests.get('http://reg.test.genome.network/gene/GN11998')` `print(res.text)`
bash	`curl -X GET "http://reg.test.genome.network/gene/GN11998"`

response:


```
{
  "@context": "http://reg.test.genome.network/schema/gene.jsonld",
  "@id": "http://reg.test.genome.network/gene/GN11998",
  "externalRecords": {
    "NCBI": {
      "@id": "http://www.ncbi.nlm.nih.gov/gene/7157",
      "id": "7157"
    },
    "HGNC": {
      "@id": "http://www.genenames.org/cgi-bin/gene_symbol_report?
hgnc_id=HGNC:11998",
      "id": "HGNC:11998",
      "symbol": "TP53",
      "name": "tumor protein p53"
    }
  },
  "names": [
    "LFS1",
    "p53"
  ]
}
```

Query or register an allele using HGVS expression

HGVS is one of the standard notations for describing alleles. Allele Registry allows for accessing and registering alleles using HGVS expressions.

Query canonical allele by HGVS expression

Canonical allele can be queried by HGVS string with the following HTTP GET request:
`http://{ServerName}/allele?hgvs={HGVS}`
This query returns responses with single allele object and status HTTP OK or responses with status HTTP NOT FOUND when given allele is not in the registry.

Example:

request: HTTP GET http://reg.test.genome.network/allele?hgvs=NC_000010.11:g.87894077C>T

ruby	`url = 'http://reg.test.genome.network' +`
	`'/allele?hgvs=NC_000010.11:g.87894077C>T'`
	`res = Net::HTTP.get_response(URI(URI.escape(url)))`
	`print res.body`

python	`hgvs = { "hgvs" : "NC_000010.11:g.87894077C>T" }`
	`res = requests.get('http://reg.test.genome.network/allele',params=hgvs)`
	`print(res.text)`

bash	`URL="http://reg.test.genome.network"`
	`URL+="/allele?hgvs=NC_000010.11:g.87894077C>T"`
	`URL=${URL//>/%3E}`
	`curl -X GET "${URL}"`

response: analogical like for allele URI

Bulk query of alleles with file containing HGVS expressions

In case of many HGVS queries the efficiency can be improved by grouping many HGVS expressions in single text file and sending it as a single HTTP POST request. The file content must be sent as a payload and the HTTP POST request must have the following syntax:

`http://{ServerName}/alleles?file=hgvs`

As a result the request will return vector of canonical allele objects in the same order as occurrences of corresponding HGVS expression in the file. In case of an error corresponding vector element is going to contain an error object instead of canonical allele object. Occurrence of an error for given HGVS expression does not influence the results of the others expressions.

`(TODO – add example)`

ruby	`url = 'http://reg.test.genome.network' +`
	`'/allele?hgvs=NC_000010.11:g.87894077C>T'`
	`# paste here code from the section 'Authentication'`
	`res = Net::HTTP.post_form(URI(URI.escape(request)))`
	`print res.body`

python	`url = 'http://reg.test.genome.network/alleles?file=hgvs`
	
	`res = requests.post(request)`
	`print(res.text)`

bash	`SHA1HEX=`sha1sum ./alleles.txt | cut -d \ -f 1` `URL="http://reg.test.genome.network/alleles?file=hgvs&sha1=${SHA1HEX}"` `curl -X GET "${URL}" -d @alleles.txt`

Bulk query of alleles with VCF file

Similar bulk query can be run for VCF file. In this case the input file must be a valid VCF file and must contain a ##contig parameter in the header for every chromosome id used in the file. Moreover each ##contig parameter should contain at least two fields named 'ID' and 'assembly'. In the current version of Allele Registry the only allowed value of the field 'assembly' is 'GRCh38'. The file content must be sent as a payload and the HTTP POST request must have the following syntax:

`http://{ServerName}/alleles?file=vcf`
`(TODO – add example)`

Register a new allele

Requests similar to those three described above can be used to register new alleles in Allele Registry. In this case the following two modifications must be made:

- the type of request should be HTTP PUT instead of HTTP GET or HTTP POST
- authentication parameters must be added

This kind of request returns the same response as corresponding HTTP GET or HTTP POST one, the only difference is that status HTTP NOT FOUND is never returned (new allele is added if not found in the registry).

Example:

request: HTTP PUT http://reg.test.genome.network/allele?hgvs=NC_000010.11:g.87894077C>T

ruby	`url = 'http://reg.test.genome.network' +`
	`'/allele?hgvs=NC_000010.11:g.87894077C>T'`
	`# paste here code from the section 'Authentication'`
	`res = Net::HTTP.put(URI(URI.escape(request)))`
	`print res.body`

python	`url = 'http://reg.test.genome.network/allele?hgvs='`
	`url += requests.utils.quote("NC_000010.11:g.87894077C>T")`
	`# paste here code from the section 'Authentication'`
	`res = requests.put(request)`
	`print(res.text)`

bash	`URL="http://reg.test.genome.network"`
	`URL+="/allele?hgvs=NC_000010.11:g.87894077C>T"`
	`URL=${URL//>/%3E}`
	`# paste here code from the section 'Authentication'`
	`curl -X PUT "${REQUEST}"`

Response – the same as in the corresponding example with HTTP GET.

Queries

All correct queries return list of matching objects in response's body and status HTTP OK. If there is no matching object, the response body contains empty list and the returned status is also HTTP OK (HTTP NOT FOUND is not used in case of queries). The HTTP addresses for querying objects depend on the object type:

- alleles – `http://{ServerName}/alleles`
- genes – `http://{ServerName}/genes`
- reference sequences – `http://{ServerName}/refseqs`

The type of query is defined by parameters added to the addresses above. Allele Registry allows only for query types described below.

Query objects by their names

Each type of object can be queried by one of his known names. This type of query can be executed by proper HTTP GET request with parameter “name”:

- alleles – `http://{ServerName}/alleles?name={name}`
- genes – `http://{ServerName}/genes?name={name}`
- reference sequences – `http://{ServerName}/refseqs?name={name}`

Remember that any of these queries may return empty list (if not found) or list containing more than one element (if the name is not unique). In all these cases the status HTTP OK is returned.

Example:

request: HTTP GET http://reg.test.genome.network/genes?name=TP53

ruby	`url = 'http://reg.test.genome.network/genes?name=TP53'`
	`res = Net::HTTP.get_response(URI(URI.escape(url)))`
	`print res.body`

python	`res = requests.get('http://reg.test.genome.network/genes?name=TP53')` `print(res.text)`
bash	`curl -X GET "http://reg.test.genome.network/genes?name=TP53"`

response:


```
[
  {
    "@context": "http://reg.test.genome.network/schema/gene.jsonld",
    "@id": "http://reg.test.genome.network/gene/GN11998",
    "externalRecords": {
      "NCBI": {
        "id": "7157",
        "@id": "http://www.ncbi.nlm.nih.gov/gene/7157"
      },
      "HGNC": {
        "id": "HGNC:11998",
        "symbol": "TP53",
        "name": "tumor protein p53",
        "@id": "http://www.genenames.org/cgi-bin/gene_symbol_report?
hgnc_id=HGNC:11998"
      }
```

```
    },
    "names": [
      "LFS1",
      "p53"
    ]
  }
]
```

Query canonical alleles by reference sequence locus

This type of query can return list of alleles defined in the context of given reference sequence and lying in particular region of this sequence. The simplest version of this query just returns all alleles defined for given reference:
`http://{ServerName}/alleles?refseq={name}`
The region of interest can be specified by adding optional parameters “begin” and “end”:
`http://{ServerName}/alleles?refseq={name}&begin={pos1}&end={pos2}`
Both “begin” and “end” parameters are optional and may be omitted. Missing “begin” parameter means “beginning of the reference sequence”, similarly missing “end” parameter means “the end of the reference sequence”.

Example:

request: HTTP GET
http://reg.test.genome.network/alleles?refseq=NM_000546.5&begin=290&end=295

ruby	`url = 'http://reg.test.genome.network' +`
	`  '/alleles?refseq=NM_000546.5&begin=290&end=295'`
	`res = Net::HTTP.get_response(URI(URI.escape(url)))`
	`print res.body`

python	`reg = { 'refseq' : 'NM_000546.5', 'begin' : 290, 'end' : 295 }`
	`res = requests.get('http://reg.test.genome.network/alleles', params=reg)`
	`print(res.text)`

bash	`URL="http://reg.test.genome.network"`
	`URL+="/alleles?refseq=NM_000546.5&begin=290&end=295"`
	`curl -X GET "${URL}"`

response:

```
[
  {
    "uri": "http://reg.test.genome.network/allele/CA000479",
    "names": [
      "SA001025",
      "SA001024",
      "NC_000017.11:g.7676388_7676390delGTT",
      "NM_000546.5:c.88_90delAAC"
    ],
    "type": "nucleotide",
    "genomicAlleles": [
      {
        "referenceSequence":"http://reg.test.genome.network/refseq/RS896675939861",
        "end": 7676390,
        "start": 7676387
      }
    ],
```

```
    "transcriptAlleles": [
      {
        "referenceSequence":"http://reg.test.genome.network/refseq/RS322512438994",
        "end": 292,
        "refAllele": "AAC",
        "start": 289
      }
    ]
  },
  {
    "uri": "http://reg.test.genome.network/allele/CA000497",
    "names": [
      "NC_000017.11:g.7676387C>T",
      "NM_000546.5:c.91G>A",
      "SA001065",
      "SA001064"
    ],
    "type": "nucleotide",
    "genomicAlleles": [
      {
        "referenceSequence":"http://reg.test.genome.network/refseq/RS896675939861",
        "end": 7676387,
        "refAllele": "C",
        "start": 7676386,
        "allele": "T"
      }
    ],
    "transcriptAlleles": [
      {
        "referenceSequence":"http://reg.test.genome.network/refseq/RS322512438994",
        "end": 293,
        "refAllele": "G",
        "start": 292,
        "allele": "A"
      }
    ]
  }
]
```


Query canonical alleles by identifiers from external records

Alleles can be also queried by some identifiers copied from external systems, like dbSNP rs number or ClinVar variation identifier. This kind of query has the following format:
`http://{ServerName}/alleles?{fieldName}={value}`
Supported values of {fieldName} are shown in the table below:

Field Name	example
ClinVar.variationId	.../alleles?ClinVar.variationId=186550
ClinVar.alleleId	.../alleles?ClinVar.alleleId=186550
ClinVar.RCV	.../alleles?ClinVar.RCV=RCV000168487
dbSNP.rs	.../alleles?dbSNP.rs=786204261

(TODO - examples)