Allele Registry

versions 1.01.xx

API specification

document version 1

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}`. The official instance of Allele Registry is currently available at the addresses http://reg.genome.network and http://reg.clinicalgenome.org. There is also a test server available at the location http://reg.test.genome.network, providing safe test environment. **All examples provided in this documentation use URLs from the test server, they must be changed to official URL in production-ready source code. The test and the production server may contain different set of alleles and identifiers of the same alleles may be different on both servers.**

Registration of new alleles requires an active account on the Allele Registry server. For the testing purposes please use the account with a login “testuser” and a password “testuser” available on the test server. To get an account on the production server please contact us at bcm.clingen@gmail.com. All other services than alleles registration are publicly available and do not require any authentication.

Sending HTTP requests

The Allele Registry API is based on HTTP requests. There are three types of HTTP requests used by this API: GET, POST and PUT. All HTTP GET requests can be send with the use of any Internet browser (just copy the URL to the address bar), but for sending POST and PUT requests some more advanced tool is needed. Moreover, all PUT requests requires authentication, what is described in the next section. Ready scripts for sending POST and PUT requests with payload from given file can be found at `http://{ServerName}/doc/scripts`.

Below you can find examples how to send GET, POST and PUT requests from bash console and from chosen programming languages. Variables “login” and “password” must be set in advance.

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.

```
# send a GET request with parameter
URL="http://reg.test.genome.network/allele?hgvs=NC_000010.11:g.87894077C>T"
URL=${URL//>/%3E}   # convert symbol > to special code %3E
curl -X GET "${URL}"

# send a POST requests with parameter and payload taken from the file alleles.txt
URL="http://reg.test.genome.network/alleles?file=hgvs"
curl -X POST "${URL}" --data-binary @alleles.txt

# calculate authentication parameters and send a PUT request
# you have to set LOGIN and PASSWORD here
URL="http://reg.test.genome.network/allele?hgvs=NC_000010.11:g.87894077del"
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=${TOKEN}"
curl -X PUT "${REQUEST}"
```

Ruby

All included ruby code snippets should work with ruby version >= 1.8.7.

```
require 'net/http'
require 'digest/sha1'

# send a GET request with parameter
url = 'http://reg.test.genome.network/allele?hgvs=NC_000010.11:g.87894077C>T'
url = URI.escape(url)   # convert symbol > to special code %3E
http = Net::HTTP.new(URI(url).host)
req = Net::HTTP::Get.new(url)
res = http.request(req)
print res.body

# send a POST requests with parameter and payload taken from the file alleles.txt
url = 'http://reg.test.genome.network/alleles?file=hgvs'
http = Net::HTTP.new(URI(url).host)
req = Net::HTTP::Post.new(url)
req.body = File.open('alleles.txt').read
res = http.request(req)
print res.body

# calculate authentication parameters and send a PUT request
# you have to set login and password here
url = 'http://reg.test.genome.network/allele?hgvs=NC_000010.11:g.87894077del'
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}"
http = Net::HTTP.new(URI(url).host)
req = Net::HTTP::Put.new(request)
res = http.request(req)
print res.body
```

Python

Python code snippets should work with python version >= 2.7. The library “request” is not a part of the Python Standard Library and probably must be installed separately (in Linux it should be available through default package manager).

```
import requests
import hashlib
import time

# send a GET request with parameter
url = 'http://reg.test.genome.network/allele?hgvs='
# convert symbol > to special code %3E
url += requests.utils.quote("NC_000010.11:g.87894077C>T")
res = requests.get(url)
print(res.text)

# send a POST requests with parameter and payload taken from the file alleles.txt
url = 'http://reg.test.genome.network/alleles?file=hgvs'
res = requests.post(url, data=open('alleles.txt').read())
print(res.text)

# calculate authentication parameters and send a PUT request
# you have to set login and password here
url = 'http://reg.test.genome.network/allele?hgvs=NC_000010.11:g.87894077del'
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
res = requests.put(request)
print(res.text)
```

Authentication

While all HTTP GET and HTTP POST requests are accepted without authentication, an active account in the 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 section above there are sample code snippets which may be used for preparing a request with authentication.

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
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)
IncorrectRequest	The request sent to the server is incorrect.	400 (Bad Request)
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)
RequestTooLarge	The request size cannot exceed 2000 records. It means that the 'limit' parameter in queries must be set to value <= 2000. Also size of any bulk requests cannot exceed 2000 alleles.	400 (Bad Request)
InternalServerError	Internal error occurred. Please, report it as an error.	500 (Internal Server Error)

Parameter set in HTTP header

All responses returned by Allele Registry have special parameter set in HTTP header:

- X-CAR-Version

It contains version of Allele Registry installed on the server. All official releases are denoted by three numbers separated by dots (e.g. 0.08.06).

Adjusting response format

As a response Allele Registry returns a single JSON object or an array of JSON objects. If URL from a HTTP request does not specify a protocol, returned JSON content is formatted in user-friendly way. When protocol is set to "json", returned JSON objects are compressed in single lines without any unnecessary white spaces. If a response consists of an array of objects, each object is saved in separate line. First line starts from '[' character, following lines start from ',' character and the last line consists of single ']' character.

The content of a response can be controlled by optional parameter "fields". It can have one of the following values:

- `fields=none[+|-field][+|-field][+|-field]…`
- `fields=all[+|-field][+|-field][+|-field]…`

The first version ("none") corresponds to empty documents to which we can add expected fields, e.g:

```
fields=none+@id+externalRecords.dbSNP.rs
```

means that returned documents are going to contain only fields "@id" and "externalRecords.dbSNP.rs". The rest of document content will be omitted. The second version of "field" parameter ("all") starts from full document structure and allow to modify content of the response by cutting of ("subtract") unwanted fields. Both these expressions are always parsed from left to right, what allows for defining more complicate rules, e.g.:

```
fields=none+externalRecords-externalRecords.dbSNP
```

is going to return documents with structure "externalRecords", but without nested structure "dbSNP".

Objects returned by API calls

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

- 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. Objects defined below are used in responses for a majority of API requests described in this document.

All objects may contain some additional fields, not described in this documentation. No assumption should be made about these undocumented fields.

Allele

URL: `http://{ServerName}/allele/{id}`

There are two type of Allele objects: “nucleotide” (in genomic space) and “amino-acid” (in protein space). Identifiers of the former ones begin with prefix “CA”, while the latter ones have prefix “PA”.

The successful response contains exactly one Allele object with the fields defined in the following table.

Allele object:

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.
externalSources	Object externalSources (see below)	only if there are links provided by users; this section is not visible by default	See “External sources” section. The parameter “fields” must be adjusted to fetch this section.
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 Allele object:

Name	**Type**	**When returned?**	**Description**
dbSNP	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to record in dbSNP • rs – rs number from dbSNP
ClinVarAlleles	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to Allele record in ClinVar • alleleId • preferredName
ClinVarVariations	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to Variation record in ClinVar • variationId • RCV – array of strings
MyVariantInfo_hg38	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to record in MyVariant.info (hg38) • id – record ID (for hg38 assembly)
MyVariantInfo_hg19	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to record in MyVariant.info (hg19) • id – record ID (for hg19 assembly)
ExAC	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to record in ExAC • id – ExAC variant ID • variant – ExAC variant name
gnomAD	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to record in gnomAD • id – gnomAD variant ID • variant – gnomAD variant name
COSMIC	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to record in COSMIC (is set only for active COSMIC records) • id – COSMIC variant ID • active – true/false

externalSources – object used in definition of Allele object:

Name	**Type**	**When returned?**	**Description**
name of the external source is used	An array of objects	Only if non-empty	Objects contain the following fields: • @id – link to record (preferably for API level) • guiUrl – link to record (preferably for GUI level, may be the same as the previous one) • guiLabel – user-friendly label

alleleDefinition – object used in definition of Allele object:

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.
gene	A gene URI	If reference sequence has assigned gene	The URI of the gene assigned to the reference sequence
geneSymbol	String	If reference sequence has assigned gene	The official symbol of the gene assigned to the reference sequence
geneNCBI_id	A non-negative integer	If reference sequence has assigned gene	The NCBI id of the gene assigned to 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
referenceGenome	“NCBI36” or “GRCh37” or “GRCh38”	If and only if the reference sequence linked above has the same field	Value of this property is taken from corresponding field in the reference sequence object with the URI given above.
chromosome	one of the strings: “1”, “2”, …, “22”, “X”, “Y”, “MT”	If and only if the reference sequence linked above has the same field	Value of this property is taken from corresponding field in the reference sequence object with the URI given above.

coordinates – object used in definition of alleleDefinition object (that is used in Allele object):

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:

request: HTTP 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
      }
    ],
    "ClinVarVariations": [
      {
        "@id": "http://www.ncbi.nlm.nih.gov/clinvar/variation/186550",
        "variationId": 186550,
        "RCV": [ "RCV000166164" ]
      }
    ],
    "ClinVarAlleles": [
      {
        "@id": "http://www.ncbi.nlm.nih.gov/clinvar/?term=183678[alleleid]",
        "alleleId": 183678,
        "preferredName": "NM_000059.3(BRCA2):c.1543A>G (p.Thr515Ala)"
      }
    ]
  },
  "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"
        }
      ],
      "referenceGenome": "GRCh38",
      "chromosome": "13"
    }
  ],
  "transcriptAlleles": [
    {
      "coordinates": [
        {
          "end": 1770,
          "allele": "G",
          "start": 1769,
          "referenceAllele": "A"
        }
      ],
      "referenceSequence": "http://reg.test.genome.network/refseq/RS938330737581",
      "gene": "http://reg.genome.network/gene/GN1101",
      "hgvs": [ "NM_000059.3:c.1543A>G", "LRG_293t1: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
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 definition:

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

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 and registration of alleles by their definition

Alleles can be unambiguously defined by providing set of parameters. One of the possibility is to describe alleles by providing identifier of reference sequence, begin and end position of the original subsequence and new sequence inserted in this region. The best option for defining variant as single label is to use HGVS notation. HGVS is the standard notations for variants definition, its description can be found at http://varnomen.hgvs.org. Allele Registry allows for querying and registering alleles by using both HGVS expressions and VCF files.

Query allele by HGVS expression

Single 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. When given allele is not in the registry, the allele object is also returned, but the field "@id" contains value "_:CA" instead of allele URI. In both cases the status HTTP SUCCESS is returned.

Example:

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

response: analogical like for an corresponding allele URI

Bulk query of alleles with HGVS expressions or identifiers

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. Each line in this text file must contain expression for single Allele. Alleles can be also queried in the same way by using CA/PA IDs or some other identifiers. The file content must be sent as a payload and the HTTP POST request must have the following syntax:

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

As a result the request will return vector of canonical allele objects in the same order as occurrences of corresponding expression in the payload. 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 single line in payload does not influence the results of the others lines.

The value of the parameter "file" defines the content of the payload. Allowed values are summarized in the following table:

{name}	Required format in the payload
hgvs	well-defined HGVS expression
id	CA ID or PA ID
MyVariantInfo_hg19.id	e.g. chr9:g.107620835G>A
MyVariantInfo_hg38.id	the same as above
ExAC.id	e.g. 5-112043382-A-G
gnomAD.id	the same as above

Example

To send HTTP POST request with payload the script `request_with_payload.sh` available at the location `http://{ServerName}/doc/scripts/` can be used. Versions for Python and Ruby are also available.

Request: request_with_payload.sh http://reg.test.genome.network/alleles?file=hgvs hgvs_3.txt

The file hgvs_3.txt contains the following 3 lines:

```
NC_000017.10:g.43009069G>C
NC_000017.10:g.43009090G>T
NC_000017.10:g.43009127delG
```

Response:

```
[
  {
    "@context": "http://reg.test.genome.network/schema/allele.jsonld",
    "@id": "http://reg.test.genome.network/allele/CA2513066",
    "type": "nucleotide",
    "externalRecords": {
      "ExAC": [ {
          "variant": "17:43009069 G / C",
          "@id": "http://exac.broadinstitute.org/variant/17-43009069-G-C"
      } ]
    },
    "genomicAlleles": [
      {
        "referenceGenome": "GRCh38",
        "chromosome": "17",
        "referenceSequence": "http://reg.test.genome.network/refseq/RS000065",
        "hgvs": [ "NC_000017.11:g.44931701G>C", "CM000679.2:g.44931701G>C" ],
        "coordinates": [ {
            "allele": "C",
            "start": 44931700,
            "end": 44931701,
            "referenceAllele": "G"
        } ]
      },
      {
        "referenceGenome": "GRCh37" ,
        "coordinates": [ {
            "start": 43009068,
            "allele": "C",
            "referenceAllele": "G",
            "end": 43009069
        } ],
        "hgvs": [ "NC_000017.10:g.43009069G>C", "CM000679.1:g.43009069G>C" ],
        "chromosome": "17",
        "referenceSequence": "http://reg.test.genome.network/refseq/RS000041"
      }
    ],
    "transcriptAlleles": [
      {
        "referenceSequence": "http://reg.test.genome.network/refseq/RS020036",
        "hgvs": [ "NM_001264573.1:c.1454C>G" ],
        "gene": "http://reg.test.genome.network/gene/GN027102",
        "coordinates": [ {
            "allele": "G",
            "start": 1551,
            "end": 1552,
            "referenceAllele": "C"
        } ]
      },
      {
        "referenceSequence": "http://reg.test.genome.network/refseq/RS394040",
        "hgvs": [ "ENST00000593135.5:c.1418C>G" ],
        "coordinates": [ {
            "allele": "G",
            "start": 1515,
            "end": 1516,
            "referenceAllele": "C"
        } ]
    } ]
  },
  {
    "@context": "http://reg.test.genome.network/schema/allele.jsonld",
    "@id": "http://reg.test.genome.network/allele/CA2513067",
    "type": "nucleotide",
    "genomicAlleles": [
      {
        "referenceGenome": "GRCh38",
        "chromosome": "17",
        "referenceSequence": "http://reg.test.genome.network/refseq/RS000065",
        "hgvs": [ "NC_000017.11:g.44931722G>T", "CM000679.2:g.44931722G>T" ],
        "coordinates": [ {
            "allele": "T",
            "start": 44931721,
            "end": 44931722,
            "referenceAllele": "G"
        } ]
      },
      {
        "referenceGenome": "GRCh37" ,
        "coordinates": [ {
            "start": 43009089,
            "allele": "T",
            "referenceAllele": "G",
            "end": 43009090
        } ],
        "hgvs": [ "NC_000017.10:g.43009090G>T", "CM000679.1:g.43009090G>T" ],
        "chromosome": "17",
        "referenceSequence": "http://reg.test.genome.network/refseq/RS000041"
      }
    ],
    "transcriptAlleles": [
      {
        "referenceSequence": "http://reg.test.genome.network/refseq/RS020036",
        "hgvs": [ "NM_001264573.1:c.1433C>A" ],
        "gene": "http://reg.test.genome.network/gene/GN027102",
        "coordinates": [ {
            "allele": "A",
            "start": 1530,
            "end": 1531,
            "referenceAllele": "C"
        } ]
      } ]
    },
    {
      "@context": "http://reg.test.genome.network/schema/allele.jsonld",
      "@id": "http://reg.test.genome.network/allele/CA2513074",
      "type": "nucleotide",
      "externalRecords": {
        "ExAC": [ {
            "variant": "17:43009126 AG / A",
            "@id": "http://exac.broadinstitute.org/variant/17-43009126-AG-A"
        } ]
      },
      "genomicAlleles": [
        {
          "referenceGenome": "GRCh38",
          "chromosome": "17",
          "referenceSequence": "http://reg.test.genome.network/refseq/RS000065",
          "hgvs": [ "NC_000017.11:g.44931762del", "CM000679.2:g.44931762del" ],
          "coordinates": [ {
              "allele": "",
              "start": 44931758,
              "end": 44931759,
              "referenceAllele": "G"
          } ]
        },
        {
          "referenceGenome": "GRCh37" ,
          "coordinates": [ {
              "start": 43009126,
              "allele": "",
              "referenceAllele": "G",
              "end": 43009127
          } ],
          "hgvs": [ "NC_000017.10:g.43009130del", "CM000679.1:g.43009130del" ],
          "chromosome": "17",
          "referenceSequence": "http://reg.test.genome.network/refseq/RS000041"
        }
      ],
      "transcriptAlleles": [
        {
          "referenceSequence": "http://reg.test.genome.network/refseq/RS020036",
          "hgvs": [ "NM_001264573.1:c.1426-30del" ],
          "gene": "http://reg.test.genome.network/gene/GN027102",
          "coordinates": [ {
              "startIntronDirection": 45,
              "allele": "",
              "start": 1523,
              "endIntronOffset": 32,
              "endIntronDirection": 45,
              "end": 1523,
              "referenceAllele": "C",
              "startIntronOffset": 33
          } ]
        }
      ]
    }
  ]
```

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'. The current version of Allele Registry support the following values of ‘ID’ and ‘assembly’ fields:

- ID: 1-22, X, Y, M, MT, chr1-chr22, chrX, chrY, chrM, chrMT
- assembly: NCBI36, GRCh37, GRCh38, hg18, hg19

‘ID’ values ‘M’ and ‘MT’ denote mitochondrial DNA. For assemblies ‘NCBI36’, ‘hg18’ and ‘hg19’, variants in mitochondrial DNA are mapped to NC_001807.4 reference sequence while for assemblies ‘GRCh37’ and ‘GRCh38’ they are mapped to NC_012920.1 reference sequence. Whole VCF file content must be sent as a payload and the HTTP POST request must have the following syntax:

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

Example

To send HTTP POST request with payload the script `request_with_payload.sh` available at the location `http://{ServerName}/doc/scripts/` can be used. Versions for Python and Ruby are also available.

Request: request_with_payload.sh http://reg.test.genome.network/alleles?file=vcf test_3.vcf

The file test_3.vcf has the following content:

```
##fileformat=VCFv4.1
##contig=<ID=chr17,assembly=GRCh37>
#CHROM      POS   ID    REF   ALT   QUAL  FILTER      INFO
chr17 43009069    .     G     C     59    PASS  DB
chr17 43009090    .     G     T     59    PASS  DB
chr17 43009126    .     AG    A     60    PASS  DB
```

Response: The same as in the example above (bulk query of alleles with HGVS expressions).

Register new alleles

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 requests, the only difference is that the field “@id” in returned allele objects has always valid URI (new allele is automatically added if not found in the registry).

Example 1:

To send HTTP PUT request with payload the script `request_with_payload.sh` available at the location `http://{ServerName}/doc/scripts/` can be used. Versions for Python and Ruby are also available. The login and password must be provided as third and fourth parameters (please see the Introduction for details).

Request:

request_with_payload.sh http://reg.test.genome.network/alleles?file=hgvs hgvs_3.txt testuser testuser

The content of file hgvs_3.txt and the response are the same as in the corresponding example with HTTP POST request (bulk query of alleles with HGVS expressions).

Example 2:

Request:

request_with_payload.sh http://reg.test.genome.network/alleles?file=vcf test_3.vcf testuser testuser

The content of file test_3.vcf and the response are the same as in the corresponding example with HTTP POST request (bulk query of alleles with VCF file).

Annotation of VCF files

Allele Registry can annotate VCF files with known variants identifiers. The VCF file must be sent as payload in POST or PUT request.

Annotation of VCF file

Allele Registry parses the content of VCF file sent as payload and returns the same VCF file with additional identifiers added to ID column. The request has the following syntax:

`http://{ServerName}/annotateVcf?assembly={assembly}&ids={ids}`

Required parameters:

- assembly – reference genome assembly, possible values: NCBI36, hg18, GRCh37, hg19, GRCh38, hg38
- ids – list of identifers to add to ID column (separated by commas), possible values: CA, dbSNP.rs, ClinVar.alleleId, ClinVar.variationId, MyVariantInfo_hg19.id, MyVariantInfo_hg38.id, ExAC.id, gnomAD.id, ClinVar.RCV, COSMIC.id.

Example:

Request: HTTP POST http://reg.test.genome.network/annotateVcf?assembly=hg19&ids=CA, dbSNP.rs

The payload must contain VCF file with variants defined on the hg19 genome assembly. The request will return the same VCF file, with CA ID and dbSNP rs ID added to ID column for all variants that are already registered in the Registry.

Annotation of VCF file with registration of unknown alleles

The request described above can be also sent as PUT request. In this case valid authentication parameters must be added to the URL. The PUT request will register automatically unknown alleles, so all recognized records in the VCF file will have assigned CA ID.

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. All queries accept two special optional parameters:

- skip – number of first records to skip (default 0)
- limit – maximal number of records to return (unlimited if not set)

Alleles are always returned in order corresponding to their position on GRCh38 genome.

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

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 single gene by HGNC symbol

The single gene can be queried by HGNC symbol.

`http://{ServerName}/gene?HGNC.symbol={symbol}`

Example:

request: HTTP GET
http://reg.test.genome.network/gene?HGNC.symbol=TP53

response:

Single gene object or HTTP NOT FOUND error.

Query all genes

The following request can be used to return the list of all genes:

`http://{ServerName}/genes`

Example:

request: HTTP GET
http://reg.test.genome.network/genes

response:

List of objects, each object corresponds to single gene.

Query reference sequences by gene

This query returns list of all reference sequences like gene regions, transcripts and proteins, assigned to given gene.

`http://{ServerName}/refseqs?gene={gene_id}`

It returns list of matched references sequences.

Example:

request: HTTP GET
http://reg.test.genome.network/refseqs?gene=GN000123

response:

List of objects, each object corresponds to single reference sequence.

Query reference sequences by reference genome

The query returns list of all reference sequences assigned to given genome build (chromosomes, alternate regions and patches).

```
http://{ServerName}/refseqs?referenceGenome={assembly}
```

The parameter “assembly” has to have one of the following values: NCBI36, GRCh37, GRCh38. The query returns list of matched references sequences.

Example:

request: HTTP GET
http://reg.test.genome.network/refseqs?referenceGenome=GRCh38

response:

List of objects, each object corresponds to single reference sequence (chromosome, mitochondrial DNA, alternate region or patch).

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

response:

List of objects, each object corresponds to single canonical allele.

Query canonical alleles by genes

Alleles can be queried by genes they are connected to. The query is called by the following HTTP GET request:

```
http://{ServerName}/alleles?gene={name}
```

It returns list of matched alleles.

Example:

request: HTTP GET
http://reg.test.genome.network/alleles?gene=TP53&limit=100
response:

List of objects, each object corresponds to single canonical allele.

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=323677`
`ClinVar.RCV`	`.../alleles?ClinVar.RCV=RCV000168487`
`dbSNP.rs`	`.../alleles?dbSNP.rs=786204261`
`MyVariantInfo_hg19.id`	`.../alleles?MyVariantInfo_hg19.id=chr14:g.23888645T>C`
`MyVariantInfo_hg38.id`	`.../alleles?MyVariantInfo_hg38.id=chr14:g.23419436T>C`
`ExAC.id`	`.../alleles?ExAC.id=14-23888645-T-C`
`gnomAD.id`	`.../alleles?gnomAD.id=14-23888645-T-C`

External sources

The goal of External Sources is to allow CAR users for publication of links to their variant-related data. Published links are returned by API and are visible on CAR's website. These entries are shown in "externalSources" section in Allele document. On the API level, this section is not returned by default. The parameter "fields" (see "Adjusting response format") must be set appropriately (e.g. fields=all) in requests returning Allele documents to fetch "externalSources" as a part of a response.

List properties of all external sources

This request returns vector of all external sources defined in the system.
Request: `GET http://{ServerName}/externalSources`

List properties of single external source

This request returns single object describing given external source.
Request: `GET http://{ServerName}/externalSource/{sourceName}`
`{sourceName}` - name of an external source

Return vector of alleles having links to given external source

This request returns vector of Allele objects.
Request: `GET /alleles?externalSource={sourceName}`
`{sourceName}` - name of an external source
Optional pagination parameters *skip* and *limit* are supported (see "Queries").

Register single link to external source

This requests is accesible only for users assigned to given external source.
Request: `PUT /allele/{CAid}/externalSource/{sourceName}`
`{CAid}` – allele ID (CA or PA ID)
`{sourceName}` - name of an external source
Depending on the definition of the external source, some of the following parameters may be required:

- `p1` – value for the first parameter
- `p2` – value for the second parameter
- `p3` – value for the third parameter

The number of parameters (p1, p2, p3) must match the definition of the external source.
The request return single Allele object with given ID or HTTP NOT FOUND error if Allele with given ID does not exist.

Bulk registration of links to external sources

This requests is accesible only for users assigned to given external source.
Request 1: `/alleles?file=id+externalSource.{sourceName}`
Request 2: `/alleles?file=hgvs+externalSource.{sourceName}`
`{sourceName}` - name of the external source
The payload must contain two columns separated by TAB character. The first column depend on the request type:

- For Request 1 (with id) it must contain CA/PA identifiers. If Allele with given ID does not exists, an error object for is returned (NotFound).
- For Request 2 (with hgvs) hgvs expressions is required. If given Allele does not exists, it is registered on the fly.

The second column must contain values of parameters required by given external source. The values must be separated by single SPACE character.
Both requests return vector of corresponding Allele objects.

Remove link(s) to external source from given allele

This requests is accesible only for users assigned to given external source.
Request: `DELETE /allele/{CAid}/externalSource/{sourceName}`
`{CAid}` – allele ID (CA or PA ID)
`{sourceName}` - name of an external source
Depending on the definition of the external source, some of the following parameters may be set:

- `p1` – value for the first parameter
- `p2` – value for the second parameter
- `p3` – value for the third parameter

If no parameters are provided, all links defined for this pair Allele/External Source are deleted. In other case, the number of parameters (p1, p2, p3) must match the definition of the external source (then single link is deleted only).
The request return single Allele object with given ID or HTTP NOT FOUND error if Allele with given ID does not exist.

Remove all links to given external source

This requests is accesible only for users assigned to given external source.
This request remove ALL links to given external source (from all Alleles).
Request: `DELETE /externalSource/{sourceName}/links`
`{sourceName}` - name of an external source
This request returns empty object.