HGVS

Release 1.3.0b2.dev1+g4d16efb

Apr 22, 2019

Contents

hgvs is a Python package to parse, format, validate, normalize, and map biological sequence variants according to recommendations of the Human Genome Variation Society. Documentation at https://hgvs.readthedocs.io/

Source | Documentation | Discuss | Issues

CHAPTER 1

Contents

1.1 Introduction

Genome, transcript, and protein sequence variants are typically reported using the variation nomenclature ("varnomen") recommendations provided by the Human Genome Variation Society (HGVS) (Taschner and den Dunnen, 2011). Most variants are deceptively simple looking, such as NM_021960.4:c.740C>T. In reality, the varnomen standard provides for much more complex concepts and representations.

As high-throughput sequencing becomes commonplace in the investigation and diagnosis of disease, it is essential that communicating variants from sequencing projects to the scientific community and from diagnostic laboratories to health care providers is easy and accurate. The HGVS mutation nomenclature recommendations are generally accepted for the communication of sequence variation: they are widely endorsed by professional organizations, mandated by numerous journals, and the prevalent representation used by databases and interactive scientific software tools. The guidelines – originally devised to standardize the representation of variants discovered before the advent of high-throughput sequencing – are now approved by the HGVS and continue to evolve under the auspices of the Human Variome Project. Unfortunately, the complexity of biological phenomena and the breadth of the varnomen standard makes it difficult to implement the standard in software, which in turn makes using the standard in high-throughput analyses difficult.

This package, *hgvs*, is an easy-to-use Python library for parsing, representing, formatting, and mapping variants between genome, transcript, and protein sequences. The current implementation handles most (but not all) of the varnomen standard for precisely defined sequence variants. The intent is to centralize the subset of HGVS variant manipulation that is routinely used in modern, high-throughput sequencing analysis.

1.1.1 Features of the hgvs Package

- **Convenient object representation**. Manipulate variants conceptually rather than by modifying text strings. Classes model HGVS concepts such as `Interval`, intronic offsets (in `BaseOffsetPosition`), uncertainty, and types of variation (`hgvs.edit`).
- **A grammar-based parser**. *hgvs* uses *a formal grammar* to parse HGVS variants rather than string partitioning or regular expression pattern matching. This makes parsing easier to understand, extend, and validate.

- **Simple variant formatting**. Object representations of variants may be turned into HGVS strings simply by printing or "stringifying" them.
- **Robust variant mapping**. The package includes tools to map variants between genome, transcript, and protein sequences (`VariantMapper` and to perform liftover between two transcript via a common reference (`Projector`). The hgvs mapper is specifically designed to reliably handl of regions reference-transcript indel discrepancy that are not covered by other tools.
- **Additional variant validation**. The package includes tools to validate variants, separate from syntactic validation provided by the grammar.
- **Extensible data sources**. Mapping and sequence data come from UTA by default, but the package includes a well-defined service interface that enables alternative data sources.
- **Extensive automated tests**. We run extensive automated tests consisting of all supported variant types on many genes for every single commit to the source code repository. Test results are displayed publicly and immediately.

Note: Some HGVS recommendations are intentionally absent. This package is primarily concerned with the subset of the VarNomen recommendations that are relevant for high-throughput sequencing. See issues for a full set of bugs and feature requests.

1.1.2 Related tools

- Mutalyzer provides a web interface to variant validation and mapping.
- Counsyl hgvs package provides functionality conceptually similar to that of the Invitae hgvs package.

1.1.3 Support

See the section *Getting Help* for information about connecting with the community, asking questions, and *filing bug reports correctly*.

1.1.4 Links

- Variation Nomenclature Recommendations
- Human Genome Variation Society (HGVS)
- Parsley, an Python wrapper for the OMeta Parser Expression Grammar (PEG)
- Universal Transcript Archive (UTA)

1.1.5 References

hgvs: A Python package for manipulating sequence variants using HGVS nomenclature: 2018 Update.

Wang M, Callenberg KM, Dalgleish R, Fedtsov A, Fox N, Freeman PJ, Jacobs KB, Kaleta P, McMurry AJ, Prlić A, Rajaraman V, Hart RK

Human Mutation. 2018

https://www.ncbi.nlm.nih.gov/pubmed/30129167

A Python package for parsing, validating, mapping and formatting sequence variants using HGVS nomenclature.

Hart RK, Rico R, Hare E, Garcia J, Westbrook J, Fusaro VA
Bioinformatics. 31(2):268-70 (2014).
https://www.ncbi.nlm.nih.gov/pubmed/25273102

Describing structural changes by extending HGVS sequence variation nomenclature.

Taschner, P. E. M., & den Dunnen, J. T.
Human Mutation, 32(5), 507–11. (2011).
http://www.ncbi.nlm.nih.gov/pubmed/21309030

A formalized description of the standard human variant nomenclature in Extended Backus-Naur Form.

Laros, J. F. J., Blavier, A., den Dunnen, J. T., & Taschner, P. E. M.
BMC Bioinformatics, 12 Suppl 4(Suppl 4), S5. (2011).
http://www.ncbi.nlm.nih.gov/pubmed/21992071

1.2 Quick Start

This tutorial provides a comprehensive example of how to use the HGVS package. Specifically, we'll:

- install hgvs
- parse a genomic variant
- project the genomic variant to all transcripts
- infer the amino acid changes for coding transcripts

We'll use rs397509113 in BRCA1. This variant is coincident with an exon in 3 coding transcripts, an intron in 2 other coding transcripts, and a non-coding transcript.

transcript (c.)	protein (p.)	comment
NM_007294.3:c.3844del	NP_009225.1:p.(Glu1282AsnfsTer25)	
NM_007297.3:c.3703del	NP_009228.2:p.(Glu1235AsnfsTer25)	
NM_007300.3:c.3844del	NP_009231.2:p.(Glu1282AsnfsTer25)	
NM_007298.3:c.788-655del	NP_009229.2:p.?	intronic variant
NM_007299.3:c.788-655del	NP_009230.2:p.?	intronic variant
NR_027676.1:n.3980del	non-coding	non-coding transcript

1.2.1 Install hgvs

For this demo, you'll obviously need *hgvs*. In a reasonably modern environment, the following should suffice:

```
$ pip install hgvs
```

More detailed installation instructions are in *Installing hgvs*.

1.2.2 Start hgvs-shell

The *hgvs* package includes an executable called `hgvs-shell`, which sets up *hgvs* for you. On the command line, type:

```
$ hgvs-shell
```

This is approximately the same thing as:

```
$ IPython
>>> from hgvs.easy import *
```

`hgvs.easy` connects to data sources and initializes commonly used objects that provide most functionality.

Note: Variant validation, normalization, and projection require access to external data, specifically exon structures, transcript alignments, and protein accessions. Right now, the only source of this data is via the UTA sister projects. When you import `hgvs.easy`, you will connect to publicly available data sources. If you want more information on the architecture of *hgvs* and UTA, see *Introduction*. See *Installing hgvs* for information about installing data sources locally for speed and privacy.

1.2.3 Parse the genomic variant

In the `hgvs-shell`, do:

```
>>> var_g = parse("NC_000017.11:g.43091687delC")
```

Note: *All* functionality in *hgvs* is provided by Python classes. `hgvs.easy` exposes common methods with functional forms also, which are used in this quick start guide. For example, `parse(...)` above actually calls `parser.parse(...), where `parser` is an instance of the *hgvs.parser.Parser* class.

Parsing a variant results in objects that represent the variant. A SequenceVariant object is comprised of an accession (`ac`), an HGVS sequence `type` (c,g,m,n,r,p), and 0 or more specific sequence changes (`posedit` – a POSition and EDIt).:

```
>>> var_g
SequenceVariant(ac=NC_000017.11, type=g, posedit=43091687del, gene=None)
```

The `posedit` is itself an object of the *hgvs.posedit.PosEdit* class:

```
>>> var_g.posedit
PosEdit(pos=43091687, edit=del, uncertain=False)
```

The `pos` (position) and `edit` attributes are also objects that can represent intervals and more complex edit operations like indels. The `uncertain` flag enables representation of HGVS uncertainty (typically with parentheses around the uncertain component). "stringifying" a variant regenerates an HGVS variant:

```
>>> str(var_g)
'NC_000017.11:g.43091687del'

>>> "This is a variant: {v}".format(v=var_g)
'This is a variant: NC_000017.11:g.43091687del'
```

And, in Python 3, stringification works in f-strings, like so:

```
> >> f"{var_g}"
'NC_000017.11:g.43091687del'
```

1.2.4 Validating and Normalizing Variants

hgvs provides functionality to validate and normalize variants:

```
>>> normalize(var_g)
SequenceVariant(ac=NC_000017.11, type=g, posedit=43091688del, gene=None)

>>> validate(var_g)
True
```

1.2.5 Projecting variants between sequences

When two sequences have alignments available in , a variant may be "projected" from one sequence to the other. `hgvs` supports projecting variants

- from g to c, n
- from c to g, n, p
- from n to c, g

The *hgvs.assemblymapper.AssemblyMapper* class provides a high-level interface to variant projection. `hgvs.easy` initializes AssemblyMapper instances for GRCh37 and GRCh37 as `am37` and `am38` respectively. For example:

```
>>> transcripts = am38.relevant_transcripts(var_g)
>>> sorted(transcripts)
['NM_007294.3', 'NM_007297.3', 'NM_007298.3', 'NM_007299.3', 'NM_007300.3', 'NR_
↪027676.1']
```

We can now project the genomic variant, `var_g`, to each of these transcripts using the `g_to_t` function, and the transcript variant to a protein sequnce using the `t_to_p` function.

```
>>> for ac in get_relevant_transcripts(var_g):
...     var_t = g_to_t(var_g, ac)
...     var_p = t_to_p(var_t)
...     print("-> " + str(var_t) + " (" + str(var_p) + ") ")
...
-> NM_007294.3:c.3844del (NP_009225.1:p.(Glu1282AsnfsTer25))
-> NM_007297.3:c.3703del (NP_009228.2:p.(Glu1235AsnfsTer25))
-> NM_007298.3:c.788-655del (NP_009229.2:p.?)
-> NM_007299.3:c.788-655del (NP_009230.2:p.?)
-> NM_007300.3:c.3844del (NP_009231.2:p.(Glu1282AsnfsTer25))
-> NR_027676.1:n.3980del (non-coding)
```

In `hgvs`, the `t` type can be either `c` or `n`. Only variants on coding sequences (`c.`) can be projected to a protein sequence. As a special case, `t_to_p` returns "non-coding" when the input variant is on a non-coding sequence.

1.3 Installing hgvs

1.3.1 Supported Platforms

hgvs is developed primarily on Ubuntu systems and has been reported to work on Mac. Other platforms and dependency versions are expected to work but have not been tested. Reports of successful operation on other platforms (and patches to enable this) are appreciated. **Python >=3.5 is now required.**

1.3.2 Install Prerequisites

hgvs currently requires PostgreSQL client libraries. On Ubuntu, try:

```
apt-get install libpq-dev
```

On a Mac with homebrew:

```
brew install postgresql
```

1.3.3 Use a virtual environment

Users are encouraged to use a virtual environment. The most basic method for this is:

```
$ python3 -m venv venv
$ source venv/bin/activate
```

Your shell prompt will change upon activation.

See this tutorial for more information about virtual environments.

1.3.4 Installing hgvs from PyPI (preferred)

Install hgvs via pip:

```
$ pip install hgvs
```

hgvs will install dependencies automatically.

1.3.5 Installing hgvs from source (for developers)

For the project at https://github.com/biocommons/hgvs.

Fetch the source code:

```
$ git clone https://github.com/<your github username>/hgvs
```

Then:

```
$ source venv/bin/activate # replace with path to your virtual env
$ cd hgvs
$ make develop
```

1.3.6 Installing SeqRepo (optional)

seqrepo provides an easy and efficient mechanism to maintain a local sequence database.

Install seqrepo:

```
$ pip install biocommons.seqrepo
```

Then, choose a file path that has at least 10GB of space available. By default, seqrepo will use /usr/local/share/serepo/. Make that directory:

```
$ mkdir /usr/local/share/seqrepo
```

Download an instance of the human sequence set:

```
$ seqrepo -r /usr/local/share/seqrepo pull
```

You can skip the -r if you use the default /usr/local/share/seqrepo/. This step will take 10-30 minutes, or more for slow connections.

As with UTA, you tell hgvs to use this feature via an environment variable:

```
$ export HGVS_SEQREPO_DIR=/usr/local/share/seqrepo/20160906
```

1.3.7 Local Installation of UTA (optional)

The easiest way to install UTA locally is to use the docker image:

$ docker run -d –name uta_20170117 -p 15032:5432 biocommons/uta:uta_20170117

If you do this, then set:

```
$ export UTA_DB_URL=postgresql://anonymous@localhost:15032/uta/uta_20170117
```

If you don't set this variable, *hgvs* will use the remote uta database.

1.3.8 Test your installation

hgvs installs *hgvs-shell*, a command line tool based on IPython. It's a convenience utility that imports and initializes frequently-used components. Try this:

```
(default-2.7) snafu$ hgvs-shell
INFO:root:Starting hgvs-shell 1.0.0a1
INFO:biocommons.seqrepo:biocommons.seqrepo 0.3.1
INFO:hgvs.dataproviders.seqfetcher:Using SeqRepo(/usr/local/share/seqrepo/master)␣
↪sequence fetching
INFO:hgvs.dataproviders.uta:connected to postgresql://anonymous:anonymous@localhost/
↪uta_dev/uta_20170117...

In [1]: v = hp.parse_hgvs_variant("NM_033089.6:c.571C>G")

In [2]: am37.c_to_g(v)
INFO:biocommons.seqrepo.fastadir.fastadir:Opening for reading: /usr/.../1472015601.
↪985206.fa.bgz
Out[2]: SequenceVariant(ac=NC_000020.10, type=g, posedit=278801C>G)

In [3]: am38.c_to_g(v)
INFO:biocommons.seqrepo.fastadir.fastadir:Opening for reading: /usr/.../1472026864.
↪4364622.fa.bgz
Out[3]: SequenceVariant(ac=NC_000020.11, type=g, posedit=298157C>G)
```

1.3.9 Package Versioning

hgvs uses semantic versioning. For a version *x.y.z*, incrementing x, y, or z denotes backward-incompatible changes, feature additions, and bug fixes respectively.

Version numbers for released code come directly from the repository tag. Therefore, PyPI version 0.1.2 corresponds exactly to the repository commit tagged as 0.1.2.

Users (i.e., non-developers) are encouraged to use the PyPI releases and to specify versions to stay within minor releases for API stability. For example, a line like:

```
hgvs>=1.0,<2
```

in setup.py or requirements.txt indicates that version 1.0 (any patch level) is required, and that future 1.x-series releases are acceptable.

1.4 Key Concepts

This section is intended for all users and provides an understanding of key concepts and components of the hgvs package.

1.4.1 Reference Sequence Types

The HGVS Recommendations provide for six types of reference sequences. Because the type influences the syntax and object representation in the hgvs package, it is important to understand these distinctions. A summary of the types follows:

Type	Sequence	Coordinates	Datum	Example
g.	DNA	Continuous	Sequence start	NC_000007.13:g.21582936G>A
m.	DNA	Continuous	Sequence start	NC_012920.1:m.8993T>C
c.	DNA	Base-Offset	Translation start	NM_001277115.1:c.351+115T>C
n.	DNA	Base-Offset	Sequence start	NM_000518.4:n.76_92del
r.	RNA	Base-Offset	Sequence start	NR_111984.1:r.44g>a
p.	AA	Continuous	Sequence start	NP_001264044.1:p.(Ala25Thr)

Datum refers to the definition for position 1 in the sequence. "Sequence start" means the first position of the sequence. "Translation start" means the position of the ATG that typically starts translation (only for coding transcripts).

Continuous coordinates are the familiar ordinal counting (1, 2, 3, . . .). There are no breaks for intervening sequence.

Base-Offset coordinates use a base position, which is an index in the specified sequence, and an optional offset from that base position. Non-zero offsets refer to non-coding sequence, such as 5' UTR, 3' UTR, or intronic position. Examples are 22 (with a zero offset), 22+6, and *6. There is no zero position; that is, the positions around the translation start are . . . , -3, -2, -1, 1, 2, 3, ...

1.4.2 Variant Object Representation

HGVS variants are represented using classes that represent elemental concepts of an HGVS sequence variant. Each of the objects contains references to data that define the objects; those data may be Python built in types such as integers

(int) or strings (unicode), or they may be other classes in the hgvs package.

For example, a variant parsed like this:

```
>>> import hgvs.parser
>>> hgvsparser = hgvs.parser.Parser()
>>> var = hgvsparser.parse_hgvs_variant('NM_001197320.1:c.281C>T')
```

will generate an object tree like the following:

Fig. 1: A typical object tree created by parsing a variant. Verticies show the property name with property type in parentheses.

For that variant, the properties may be obtained easily by dot lookup:

```
>>> var.ac
'NM_001197320.1'
>>> var.type
'c'
>>> var.posedit
PosEdit(pos=281, edit=C>T, uncertain=False)
>>> var.posedit.pos
BaseOffsetInterval(start=281, end=281, uncertain=False)
>>> var.posedit.pos.start, var.posedit.pos.end
(BaseOffsetPosition(base=281, offset=0, datum=Datum.CDS_START, uncertain=False),
 BaseOffsetPosition(base=281, offset=0, datum=Datum.CDS_START, uncertain=False))
>>> var.posedit.edit
NARefAlt(ref='C', alt='T', uncertain=False)
```

The object representation makes it easy to modify variants conceptually rather than textually. For example, if the previous variant was inferred rather than sequenced, we might wish to declare that it is uncertain, which then causes the stringified version to contain the edit in parentheses:

```
>>> var.posedit.uncertain = True
>>> str(var)
'NM_001197320.1:c.(281C>T)'
```

1.4.3 Variant Mapping Tools

Variant mapping is supported by several modules. Most users will likely be content with `hgvs.variant.AssemblyMapper`. For completeness, it may help to understand how all of the mappers relate to each other.

`hgvs.alignmentmapper.AlignmentMapper`

The AlignmentMapper uses CIGAR to map pairs of exon segments (typically exons in the transcript and genomic sequences). It is must be instantiated with a transcript accession, reference accession, and alignment method, and provides functions to map sequence intervals (not variants) for the specified alignment. It is also accommodates strand orientation.

`hgvs.variantmapper.VariantMapper`

The VariantMapper uses *`hgvs.alignmentmapper.AlignmentMapper`* to provide g<->r, r<->c, g<->c, and c->p transformations for `SequenceVariant` objects. As with the AlignmentMapper, it must be instantiated with an appropriate transcript, reference, and alignment method.

`hgvs.assemblymapper.AssemblyMapper`

VariantMapper requires that the caller provide a transcript accession and an appropriate reference sequence, which in turn requires knowing the correct reference sequence. The alignment method is also required. While the VariantMapper interface serves the general case of mapping to any sequence (including patch sequences), it is burdensome for the most common case. AssemblyMapper wraps VariantMapper to provide identical mapping functionality that is tailored for mapping between a transcript and a primary assembly.

hgvs.projector.Projector

Projector maps variants between transcripts using a common reference and alignment method. For example, this tool can transfer a variant from one RefSeq to another, or even from an Ensembl transcript to a RefSeq.

Fig. 2: Mapping tools available in the hgvs package. r1 is a genomic reference (*e.g.,* NC_000014.8). t1 and t2 are transcripts (*e.g.,* NM_000551.2). p1 is a protein sequence (*e.g.,* NP_012345.6).

1.4.4 External Data Sources

Variant mapping and validation requires access to external data, specifically exon structures, transcript alignments, accessions, and sequences. In order to isolate the hgvs package from the myriad choices and tradeoffs, these data are provided through an implementation of the (abstract) Data Provider Interface (*hgvs.dataproviders.interface*). Currently, the only concrete implementation of the data provider interface uses UTA, an archive of transcripts, transcript sequences, and transcript-reference sequence alignments.

Invitae provides a public UTA instance at `uta.biocommons.org:5432` (PostgreSQL). *hgvs* uses this public UTA instance by default, so most users won't need to worry about this aspect of the hgvs package. However, a docker image of UTA is also available; see *Installing hgvs* for details.

Alternatively, users may implement their own providers that conform to the data providers interface. See *hgvs.dataproviders.uta* for an example.

1.5 Examples

The following examples are derived directly from IPython notebooks in the hgvs source code examples directory.

1.5.1 Creating a SequenceVariant from scratch

0. Overview

A SequenceVariant consists of an accession (a string), a sequence type (a string), and a PosEdit, like this:

var = hgvs.sequencevariant.SequenceVariant(ac='NM_01234.5', type='c', posedit=...)

Unsurprisingly, a PosEdit consists of separate position and Edit objects. A position is generally an Interval, which in turn is comprised of SimplePosition or BaseOffsetPosition objects. An edit is a subclass of Edit, which includes classes like NARefAlt for substitutions, deletions, and insertions) and Dup (for duplications).

Importantly, each of the objects we're building has a rule in the parser, which means that you have the tools to serialize and deserialize (parse) each of the components that we're about to construct.

1. Make an Interval to define a position of the edit

```
import hgvs.location
import hgvs.posedit
```

```
start = hgvs.location.BaseOffsetPosition(base=200,offset=-6,datum=hgvs.location.Datum.
↪CDS_START)
start, str(start)
```

```
(BaseOffsetPosition(base=200, offset=-6, datum=Datum.CDS_START, uncertain=False),
 '200-6')
```

```
end = hgvs.location.BaseOffsetPosition(base=22,datum=hgvs.location.Datum.CDS_END)
end, str(end)
```

```
(BaseOffsetPosition(base=22, offset=0, datum=Datum.CDS_END, uncertain=False),
 '*22')
```

```
iv = hgvs.location.Interval(start=start,end=end)
iv, str(iv)
```

```
(Interval(start=200-6, end=*22, uncertain=False), '200-6_*22')
```

2. Make an edit object

```
import hgvs.edit, hgvs.posedit
```

```
edit = hgvs.edit.NARefAlt(ref='A',alt='T')
edit, str(edit)
```

```
(NARefAlt(ref='A', alt='T', uncertain=False), 'A>T')
```

```
posedit = hgvs.posedit.PosEdit(pos=iv,edit=edit)
posedit, str(posedit)
```

```
(PosEdit(pos=200-6_*22, edit=A>T, uncertain=False), '200-6_*22A>T')
```

3. Make the variant

```
import hgvs.sequencevariant
```

```
var = hgvs.sequencevariant.SequenceVariant(ac='NM_01234.5', type='c', posedit=posedit)
var, str(var)
```

```
(SequenceVariant(ac=NM_01234.5, type=c, posedit=200-6_*22A>T),
 'NM_01234.5:c.200-6_*22A>T')
```

Important: It is possible to bogus variants with the hgvs package. For example, the above interval is incompatible with a SNV. See hgvs.validator.Validator for validation options.

4. Update your variant

The stringification happens on-the-fly. That means that you can update components of the variant and see the effects immediately.

```
import copy
```

```
var2 = copy.deepcopy(var)
var2.posedit.pos.start.base=456
str(var2)
```

```
'NM_01234.5:c.456-6_*22A>T'
```

```
var2 = copy.deepcopy(var)
var2.posedit.edit.alt='CT'
str(var2)
```

```
'NM_01234.5:c.200-6_*22delinsCT'
```

```
var2 = copy.deepcopy(var)
str(var2)
```

```
'NM_01234.5:c.200-6_*22A>T'
```

1.5.2 Manuscript Example

```
import hgvs
hgvs.__version__
```

```
'0.3dev-283858cb6466'
```

Parse an HGVS string into a Python structure

```
import hgvs.parser
hp = hgvs.parser.Parser()
var_c1 = hp.parse_hgvs_variant('NM_182763.2:c.688+403C>T')
var_c1, var_c1.posedit.pos.start
```

```
(SequenceVariant(ac=NM_182763.2, type=c, posedit=688+403C>T),
 BaseOffsetPosition(base=688, offset=403, datum=1, uncertain=False))
```

Open the UTA public data source for mapping and validation

```
import hgvs.dataproviders.uta
hdp = hgvs.dataproviders.uta.connect()
```

Project transcript variant NM_182763.2:c.688+403C>T to GRCh37 primary assembly using splign alignments

```
import hgvs.variantmapper
vm = hgvs.variantmapper.AssemblyMapper(
    hdp, assembly_name='GRCh37', alt_aln_method='splign')
var_g = vm.c_to_g(var_c1)
var_g
```

```
SequenceVariant(ac=NC_000001.10, type=g, posedit=150550916G>A)
```

Project genomic variant to a new transcript

```
vm.relevant_transcripts(var_g)
```

```
['NM_182763.2', 'NM_021960.4', 'NM_001197320.1']
```

```
var_c2 = vm.g_to_c(var_g,'NM_001197320.1')
var_c2
```

```
SequenceVariant(ac=NM_001197320.1, type=c, posedit=281C>T)
```

Infer protein changes for these transcript variants

```
var_p1 = vm.c_to_p(var_c1)
var_p2 = vm.c_to_p(var_c2)
var_p1, var_p2
```

```
(SequenceVariant(ac=NP_877495.1, type=p, posedit=?),
 SequenceVariant(ac=NP_001184249.1, type=p, posedit=(Ser94Phe)))
```

Format the results by “stringification”

```
print("""mapped {var_c1} ({var_p1})
    to {var_c2} ({var_p2})
   via {var_g}""".format(
        var_c1=var_c1, var_p1=var_p1,
        var_c2=var_c2, var_p2=var_p2,
        var_g=var_g))
```

```
mapped NM_182763.2:c.688+403C>T (NP_877495.1:p.?)
    to NM_001197320.1:c.281C>T (NP_001184249.1:p.(Ser94Phe))
   via NC_000001.10:g.150550916G>A
```

Validate a variant

```
import hgvs.validator
import hgvs.exceptions
vr = hgvs.validator.Validator(hdp=hdp)
try:
    vr.validate( hp.parse_hgvs_variant('NM_001197320.1:c.281C>T') )
    vr.validate( hp.parse_hgvs_variant('NM_001197320.1:c.281A>T') )
except hgvs.exceptions.HGVSError as e:
    print(e)
```

```
NM_001197320.1:c.281A>T: Variant reference does not agree with reference sequence
```

1.5.3 Automated liftover of NM_001261456.1:c.1762A>G (rs509749) to NM_001261457.1 via GRCh37

Automatically project variant from one transcript to another via common reference.

http://www.ncbi.nlm.nih.gov/projects/SNP/snp_ref.cgi?rs=509749

```
import hgvs.parser
hgvsparser = hgvs.parser.Parser()
var_c1 = hgvsparser.parse_hgvs_variant('NM_001261456.1:c.1762A>G')
```

```
import hgvs.dataproviders.uta
hdp = hgvs.dataproviders.uta.connect()
```

```
import hgvs.projector
pj = hgvs.projector.Projector(hdp=hdp,
                              alt_ac='NC_000001.10',
                              src_ac=var_c1.ac,
                              dst_ac='NM_001261457.1')
```

```
pj.project_variant_forward(var_c1)
```

```
SequenceVariant(ac=NM_001261457.1, type=c, posedit=1534A>G)
```

1.5.4 Manual liftover of NM_001261456.1:c.1762A>G (rs509749) to NM_001261457.1 via GRCh37

http://www.ncbi.nlm.nih.gov/projects/SNP/snp_ref.cgi?rs=509749

```
import hgvs.dataproviders.uta
import hgvs.variantmapper
import hgvs.parser
```

```
hdp = hgvs.dataproviders.uta.connect()
variantmapper = hgvs.variantmapper.VariantMapper(hdp)
hgvsparser = hgvs.parser.Parser()
```

```
var_c1 = hgvsparser.parse_hgvs_variant('NM_001261456.1:c.1762A>G')
var_p1 = variantmapper.c_to_p(var_c1, None)
var_c1, var_p1
```

```
(SequenceVariant(ac=NM_001261456.1, type=c, posedit=1762A>G),
 SequenceVariant(ac=MD5_e999a940ca422ec8cab9bc3cc64e0d7d, type=p,␣
↪posedit=(Met588Val)))
```

```
var_g = variantmapper.c_to_g(var_c1,'NC_000001.10')
var_g
```

```
SequenceVariant(ac=NC_000001.10, type=g, posedit=160793560A>G)
```

```
txs = hdp.get_tx_for_gene('LY9')
txs
```

```
[['LY9', 30, 1998, 'ENST00000263285', 'NC_000001.10', 'genebuild'],
 ['LY9', 1, 583, 'ENST00000368039', 'NC_000001.10', 'genebuild'],
 ['LY9', 0, 1648, 'ENST00000392203', 'NC_000001.10', 'genebuild'],
 ['LY9', 0, 1833, 'ENST00000368037', 'NC_000001.10', 'genebuild'],
 ['LY9', 211, 1024, 'ENST00000368035', 'NC_000001.10', 'genebuild'],
 ['LY9', 50, 1616, 'ENST00000341032', 'NC_000001.10', 'genebuild'],
 ['LY9', 170, 1751, 'ENST00000368041', 'NC_000001.10', 'genebuild'],
 ['LY9', 1094, 1907, 'ENST00000368040', 'NC_000001.10', 'genebuild'],
 ['LY9', 114, 2040, 'NM_001261456.1', 'AC_000133.1', 'splign'],
 ['LY9', 114, 2040, 'NM_001261456.1', 'NC_000001.10', 'blat'],
 ['LY9', 114, 2040, 'NM_001261456.1', 'NC_000001.10', 'splign'],
 ['LY9', 114, 2040, 'NM_001261456.1', 'NC_018912.2', 'splign'],
 ['LY9', 114, 696, 'NM_001033667.2', 'AC_000133.1', 'splign'],
 ['LY9', 114, 696, 'NM_001033667.2', 'NC_000001.10', 'blat'],
 ['LY9', 114, 696, 'NM_001033667.2', 'NC_000001.10', 'splign'],
 ['LY9', 114, 696, 'NM_001033667.2', 'NC_018912.2', 'splign'],
 ['LY9', 114, 2082, 'NM_002348.3', 'AC_000133.1', 'splign'],
 ['LY9', 114, 2082, 'NM_002348.3', 'NC_000001.10', 'blat'],
 ['LY9', 114, 2082, 'NM_002348.3', 'NC_000001.10', 'splign'],
 ['LY9', 114, 2082, 'NM_002348.3', 'NC_018912.2', 'splign'],
 ['LY9', 114, 1812, 'NM_001261457.1', 'AC_000133.1', 'splign'],
 ['LY9', 114, 1812, 'NM_001261457.1', 'NC_000001.10', 'blat'],
 ['LY9', 114, 1812, 'NM_001261457.1', 'NC_000001.10', 'splign'],
 ['LY9', 114, 1812, 'NM_001261457.1', 'NC_018912.2', 'splign']]
```

```
var_c2 = variantmapper.g_to_c(var_g,'NM_001261457.1',alt_aln_method='splign')
var_p2 = variantmapper.c_to_p(var_c2, None)
var_c2, var_p2
```

```
(SequenceVariant(ac=NM_001261457.1, type=c, posedit=1534A>G),
 SequenceVariant(ac=MD5_921ebefe79bff479f4bfa17e133fc084, type=p,␣
↪posedit=(Met512Val)))
```

1.5.5 Using hgvs

This notebook demonstrates major features of the hgvs package.

```
import hgvs
hgvs.__version__
```

```
'0.5.0a6.dev3+nf998c16a46b3.d20161012'
```

Variant I/O

Initialize the parser

```
# You only need to do this once per process
import hgvs.parser
hp = hgvsparser = hgvs.parser.Parser()
```

Parse a simple variant

```
v = hp.parse_hgvs_variant("NC_000007.13:g.21726874G>A")
```

```
v
```

```
SequenceVariant(ac=NC_000007.13, type=g, posedit=21726874G>A)
```

```
v.ac, v.type
```

```
('NC_000007.13', 'g')
```

```
v.posedit
```

```
PosEdit(pos=21726874, edit=G>A, uncertain=False)
```

```
v.posedit.pos
```

```
Interval(start=21726874, end=21726874, uncertain=False)
```

```
v.posedit.pos.start
```

```
SimplePosition(base=21726874, uncertain=False)
```

Parsing complex variants

```
v = hp.parse_hgvs_variant("NM_003777.3:c.13552_*36del57")
```

```
v.posedit.pos.start, v.posedit.pos.end
```

```
(BaseOffsetPosition(base=13552, offset=0, datum=1, uncertain=False),
 BaseOffsetPosition(base=36, offset=0, datum=2, uncertain=False))
```

```
v.posedit.edit
```

```
NARefAlt(ref=57, alt=None, uncertain=False)
```

Formatting variants

All objects may be formatted simply by "stringifying" or printing them using `str`, `print()`, or `"".format()`.

```
str(v)
```

```
'NM_003777.3:c.13552_*36del57'
```

```
print(v)
```

```
NM_003777.3:c.13552_*36del57
```

```
"{v} spans the CDS end".format(v=v)
```

```
'NM_003777.3:c.13552_*36del57 spans the CDS end'
```

Projecting variants between sequences

Set up a dataprovider

Mapping variants requires exon structures, alignments, CDS bounds, and raw sequence. These are provided by a `hgvs.dataprovider` instance. The only dataprovider provided with hgvs uses UTA. You may write your own by subsclassing hgvs.dataproviders.interface.

```
import hgvs.dataproviders.uta
hdp = hgvs.dataproviders.uta.connect()
```

Initialize mapper classes

The VariantMapper class projects variants between two sequence accessions using alignments from a specified source. In order to use it, you must know that two sequences are aligned. VariantMapper isn't demonstrated here.

AssemblyMapper builds on VariantMapper and handles identifying appropriate sequences. It is configured for a particular genome assembly.

```
import hgvs.variantmapper
#vm = variantmapper = hgvs.variantmapper.VariantMapper(hdp)
am37 = easyvariantmapper = hgvs.variantmapper.AssemblyMapper(hdp, assembly_name=
↪'GRCh37')
am38 = easyvariantmapper = hgvs.variantmapper.AssemblyMapper(hdp, assembly_name=
↪'GRCh38')
```

c_to_g

This is the easiest case because there is typically only one alignment between a transcript and the genome. (Exceptions exist for pseudoautosomal regions.)

```
var_c = hp.parse_hgvs_variant("NM_015120.4:c.35G>C")
var_g = am37.c_to_g(var_c)
var_g
```

```
am38.c_to_g(var_c)
```

g_to_c

In order to project a genomic variant onto a transcript, you must tell the AssemblyMapper which transcript to use.

```
am37.relevant_transcripts(var_g)
```

```
['NM_015120.4']
```

```
am37.g_to_c(var_g, "NM_015120.4")
```

```
SequenceVariant(ac=NM_015120.4, type=c, posedit=35T>C)
```

c_to_p

```
var_p = am37.c_to_p(var_c)
str(var_p)
```

```
'NP_055935.4:p.(Leu12Pro)'
```

```
var_p.posedit.uncertain = False
str(var_p)
```

```
'NP_055935.4:p.Leu12Pro'
```

Projecting in the presence of a genome-transcript gap

As of Oct 2016, 1033 RefSeq transcripts in 433 genes have gapped alignments. These gaps require special handlingin order to maintain the correspondence of positions in an alignment. hgvs uses the precomputed alignments in UTA to correctly project variants in exons containing gapped alignments.

This example demonstrates projecting variants in the presence of a gap in the alignment of NM_015120.4 (ALMS1) with GRCh37 chromosome 2. (The alignment with GRCh38 is similarly gapped.) Specifically, the adjacent genomic positions 73613031 and 73613032 correspond to the non-adjacent CDS positions 35 and 39.

```
NM_015120.4  c         15 >                          >       58
NM_015120.4  n        126 > CCGGGCGAGCTGGAGGAGGAGGAG  >      169
                           ||||||||||    ||||||||||  21=3I20=
NC_000002.11 g   73613021 > CCGGGCGAGCT---GGAGGAGGAG  > 73613041
NC_000002.11 g   73613021 < GGCCCGCTCGA---CCTCCTCCTC  < 73613041
```

```
str(am37.c_to_g(hp.parse_hgvs_variant("NM_015120.4:c.35G>C")))
```

```
'NC_000002.11:g.73613031T>C'
```

```
str(am37.c_to_g(hp.parse_hgvs_variant("NM_015120.4:c.39G>C")))
```

```
'NC_000002.11:g.73613032G>C'
```

Normalizing variants

In hgvs, normalization means shifting variants 3' (as requried by the HGVS nomenclature) as well as rewriting variants. The variant "NM_001166478.1:c.30_31insT" is in a poly-T run (on the transcript). It should be shifted 3' and is better written as dup, as shown below:

```
                                        *                       NC_000006.11:g.
↪49917127dupA
  NC_000006.11 g   49917117 > AGAAAGAAAAATAAAACAAAG  > 49917137
  NC_000006.11 g   49917117 < TCTTTCTTTTTATTTTGTTTC  < 49917137
                             ||||||||||||||||||||  21=
NM_001166478.1 n         41 < TCTTTCTTTTTATTTTGTTTC  <        21 NM_001166478.1:n.
↪35dupT
NM_001166478.1 c         41 <                        <        21 NM_001166478.1:c.30_
↪31insT
```

```
import hgvs.normalizer
hn = hgvs.normalizer.Normalizer(hdp)
```

```
v = hp.parse_hgvs_variant("NM_001166478.1:c.30_31insT")
str(hn.normalize(v))
```

```
'NM_001166478.1:c.35dupT'
```

A more complex normalization example

This example is based on https://github.com/biocommons/hgvs/issues/382/.

```
  NC_000001.11 g   27552104 > CTTCACACGCATCCTGACCTTG > 27552125
  NC_000001.11 g   27552104 < GAAGTGTGCGTAGGACTGGAAC < 27552125
                             ||||||||||||||||||||| 22=
NM_001029882.3 n        843 < GAAGTGTGCGTAGGACTGGAAC <      822
NM_001029882.3 c         12 <                        <      -10
                                      ^^
                                      NM_001029882.3:c.1_2del
                                      NM_001029882.3:n.832_833delAT
                                      NC_000001.11:g.27552114_27552115delAT
```

```
am38.c_to_g(hp.parse_hgvs_variant("NM_001029882.3:c.1A>G"))
```

```
SequenceVariant(ac=NC_000001.11, type=g, posedit=27552115T>C)
```

```
am38.c_to_g(hp.parse_hgvs_variant("NM_001029882.3:c.2T>G"))
```

```
SequenceVariant(ac=NC_000001.11, type=g, posedit=27552114A>C)
```

```
am38.c_to_g(hp.parse_hgvs_variant("NM_001029882.3:c.1_2del"))
```

```
SequenceVariant(ac=NC_000001.11, type=g, posedit=27552114_27552115delAT)
```

The genomic coordinates for the SNVs at c.1 and c.2 match those for the del at c.1_2. Good!

Now, notice what happens with c.1_3del, c.1_4del, and c.1_5del:

```
am38.c_to_g(hp.parse_hgvs_variant("NM_001029882.3:c.1_3del"))
```

```
SequenceVariant(ac=NC_000001.11, type=g, posedit=27552114_27552116delATC)
```

```
am38.c_to_g(hp.parse_hgvs_variant("NM_001029882.3:c.1_4del"))
```

```
SequenceVariant(ac=NC_000001.11, type=g, posedit=27552112_27552115delGCAT)
```

```
am38.c_to_g(hp.parse_hgvs_variant("NM_001029882.3:c.1_5del"))
```

```
SequenceVariant(ac=NC_000001.11, type=g, posedit=27552112_27552116delGCATC)
```

Explanation:

On the transcript, c.1_2delAT deletes AT from ...AGGATGCG..., resulting in ...AGGGCG.... There's no ambiguity about what sequence was actually deleted.

c.1_3delATG deletes ATG, resulting in ...AGGCG.... Note that you could also get this result by deleting GAT. This is an example of an indel that is subject to normalization and hgvs does this.

c.1_4delATGC and 1_5delATGCG have similar behaviors.

Normalization is always 3' with respect to the reference sequence. So, after projecting from a - strand transcript to the genome, normalization will go in the opposite direction to the transcript. It will have roughly the same effect as being 5' shifted on the transcript (but revcomp'd).

For more precise control, see the `normalize` and `replace_reference` options of `AssemblyMapper`.

Validating variants

`hgvs.validator.Validator` is a composite of two classes, `hgvs.validator.IntrinsicValidator` and `hgvs.validator.ExtrinsicValidator`. Intrinsic validation evaluates a given variant for *internal* consistency, such as requiring that insertions specify adjacent positions. Extrinsic validation evaluates a variant using external data, such as ensuring that the reference nucleotide in the variant matches that implied by the reference sequence and position. Validation returns `True` if successful, and raises an exception otherwise.

```
import hgvs.validator
hv = hgvs.validator.Validator(hdp)
```

```
hv.validate(hp.parse_hgvs_variant("NM_001166478.1:c.30_31insT"))
```

```
True
```

```
from hgvs.exceptions import HGVSError

try:
    hv.validate(hp.parse_hgvs_variant("NM_001166478.1:c.30_32insT"))
except HGVSError as e:
    print(e)
```

```
insertion length must be 1
```

1.6 Reference Manual

1.6.1 Grammar

Grammar Overview

Note: This section is being written.

Provide an overview of the grammar rules Also consider a document link to the grammar itself

---

HGVS Railroad Diagram

Generated from hgvs (https://github.com/biocommons/hgvs)
1b1f788ef473+ default tip
See the source code for the OMeta-based grammar

Variants

Intervals

Localized Edits

Positions

Edits (sequence changes)

Sequences

Residues

Remaining rules

HGVS Railroad Diagram

Generated from hgvs (https://github.com/biocommons/hgvs)
1b1f788ef473+ default tip
See the source code for the OMeta-based grammar

Variants

Intervals

Localized Edits

Positions

Edits (sequence changes)

Sequences

Residues

Remaining rules

1.6.2 Modules

Module Overview

Module	Classes	Description
Variant Object Representation		
hgvs.edit	hgvs.edit.AAExt hgvs.edit.AAFs hgvs.edit.AARefAlt hgvs.edit.AASub hgvs.edit.Dup hgvs.edit.Edit hgvs.edit.NACopy hgvs.edit.NADupN hgvs.edit.NARefAlt hgvs.edit.Repeat	hgvs.edit classes implement various kinds of sequence edits. For nucleic acids, these edits are independent of location; amino acids edits currently contain the location.
hgvs.hgvsposition	hgvs.hgvsposition.HGVSPosition	A non-standard representation of a sequence location without an edit. For example, NM_012345.6:c.72+5_73-2.
hgvs.location	hgvs.location.AAPosition hgvs.location.BaseOffsetPosition hgvs.location.Interval hgvs.location.SimplePosition	Various kinds of locations. Interval is a span from start to end; the others are points in a sequence.
hgvs.posedit	hgvs.posedit.PosEdit	A position+edit (really, an interval and edit).
hgvs.variant	hgvs.variant.SequenceVariant	A sequence variant of any type (g, c, m, r, n, p). A SequenceVariant is returned by hgvs.parser.Parser, and it is the input and output type for hgvs.variantmapper.VariantMapper operations.
Parsing and Formatting		
hgvs.parser	hgvs.parser.Parser	
Coordinate, Interval, and Variant Mapping/Transformation		
hgvs.projector	hgvs.projector.Projector	
hgvs.alignmentmapper	hgvs.alignmentmapper.AlignmentMapper	
hgvs.variantmapper	hgvs.variantmapper.	

Top-level module

hgvs is a package to parse, format, and manipulate biological sequence variants. See https://github.com/biocommons/hgvs/ for details.

Example use:

```
>>> import hgvs.dataproviders.uta
>>> import hgvs.parser
>>> import hgvs.variantmapper
```

start with these variants as strings >>> hgvs_g, hgvs_c = "NC_000007.13:g.36561662C>T", "NM_001637.3:c.1582G>A"

parse the genomic variant into a Python structure >>> hp = hgvs.parser.Parser() >>> var_g = hp.parse_hgvs_variant(hgvs_g) >>> var_g SequenceVariant(ac=NC_000007.13, type=g, posedit=36561662C>T, gene=None)

SequenceVariants are composed of structured objects, e.g., >>> var_g.posedit.pos.start SimplePosition(base=36561662, uncertain=False)

format by stringification >>> str(var_g) 'NC_000007.13:g.36561662C>T'

initialize the mapper for GRCh37 with splign-based alignments >>> hdp = hgvs.dataproviders.uta.connect() >>> am = hgvs.assemblymapper.AssemblyMapper(hdp, ... assembly_name="GRCh37", alt_aln_method="splign", ... replace_reference=True)

identify transcripts that overlap this genomic variant >>> transcripts = am.relevant_transcripts(var_g) >>> sorted(transcripts) ['NM_001177506.1', 'NM_001177507.1', 'NM_001637.3']

map genomic variant to one of these transcripts >>> var_c = am.g_to_c(var_g, "NM_001637.3") >>> var_c SequenceVariant(ac=NM_001637.3, type=c, posedit=1582G>A, gene=None) >>> str(var_c) 'NM_001637.3:c.1582G>A'

CDS coordinates use BaseOffsetPosition to support intronic offsets >>> var_c.posedit.pos.start BaseOffsetPosition(base=1582, offset=0, datum=Datum.CDS_START, uncertain=False)

Configuration

hgvs.config

The hgvs package uses a single, package-wide configuration instance to control package behavior. The hgvs.config module provides that configuration instance, via the *hgvs.global_config* variable.

You should not import hgvs.config directly.

Config are read from an ini-format file. *hgvs.config* implements a thin wrapper on the ConfigParser instance in order to provide *attribute* based lookups (rather than key). It also returns heuristically typed values (e.g., "True" becomes True).

Although keys are settable, they are stringified on setting and type-inferred on getting, which means that round-tripping works only for str, int, and boolean.

```
>>> import hgvs.config
```

hgvs.config.**global_config**
: Package-wide ("global") configuration, initialized with package defaults. Setting configuration in this object will change global behavior of the hgvs package.

  global_config, an instance of :ref:hgvs.config.Config, supports reading ini-like files that updates

class hgvs.config.**Config**(*extended_interpolation=True*)
Bases: object

provides an attribute-based lookup of configparser sections and settings.

read_stream(*flo*)
read configuration from ini-formatted file-like object

class hgvs.config.**ConfigGroup**(*section*)
Bases: object

The defaults are:

```
[mapping]
alt_aln_method = splign
assembly = GRCh38
in_par_assume = X
inferred_p_is_uncertain = True
normalize = True
replace_reference = True

[formatting]
p_3_letter = True
p_term_asterisk = False

[normalizer]
cross_boundaries = False
shuffle_direction = 3
validate = True
window_size = 20

[lru_cache]
maxsize = 100
```

Variant Object Representation

hgvs.edit

Representation of edit operations in HGVS variants

NARefAlt and AARefAlt are abstractions of several major variant types. They are distinguished by whether the ref and alt elements of the structure. The HGVS grammar for NA and AA are subtly different (e.g., the ref AA in a protein substitution is part of the location).

class hgvs.edit.**AAExt**(*ref=None*, *alt=None*, *aaterm=None*, *length=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

aaterm

alt

format(*conf=None*)

length

ref

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**AAFs**(*ref=None*, *alt=None*, *length=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

alt

format(*conf=None*)

length

ref

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**AARefAlt**(*ref=None*, *alt=None*, *uncertain=False*, *init_met=False*)
Bases: *hgvs.edit.Edit*

alt

format(*conf=None*)

init_met

ref

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**AASub**(*ref=None*, *alt=None*, *uncertain=False*, *init_met=False*)
Bases: *hgvs.edit.AARefAlt*

format(*conf=None*)

type
return the type of this Edit

Returns edit type (str)

class hgvs.edit.**Conv**(*from_ac=None*, *from_type=None*, *from_pos=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

Conversion

from_ac

from_pos

from_type

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**Dup**(*ref=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

format(*conf=None*)

ref

ref_s
returns a string representing the ref sequence, if it is not None and smells like a sequence

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**Edit**
Bases: object

format(*conf=None*)

class hgvs.edit.**Inv**(*ref=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

Inversion

ref

ref_n
returns an integer, either from the *seq* instance variable if it's a number, or None otherwise

ref_s

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**NACopy**(*copy=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

Represent copy number variants (Invitae-specific use)

This class is intended for Invitae use only and does not represent a standard HGVS concept. The class may be changed, moved, or removed without notice.

copy

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**NARefAlt**(*ref=None*, *alt=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

represents substitutions, deletions, insertions, and indels.

Variables

- ***ref*** – reference sequence or length
- ***alt*** – alternate sequence
- ***uncertain*** – boolean indicating whether the variant is uncertain/undetermined

alt

format(*conf=None*)

ref

ref_n
returns an integer, either from the *ref* instance variable if it's a number, or the length of ref if it's a string, or None otherwise

```
>>> NARefAlt("ACGT").ref_n
4
>>> NARefAlt("7").ref_n
7
>>> NARefAlt(7).ref_n
7
```

ref_s
returns a string representing the ref sequence, if it is not None and smells like a sequence

```
>>> NARefAlt("ACGT").ref_s
u'ACGT'
>>> NARefAlt("7").ref_s
>>> NARefAlt(7).ref_s
```

type
return the type of this Edit

Returns edit type (str)

uncertain

class hgvs.edit.**Repeat**(*ref=None*, *min=None*, *max=None*, *uncertain=False*)
Bases: *hgvs.edit.Edit*

format(*conf=None*)

max

min

ref

type
return the type of this Edit

Returns edit type (str)

uncertain

hgvs.hgvsposition

Represent partial HGVS tags that refer to a position without alleles

class hgvs.hgvsposition.**HGVSPosition**(*ac*, *type*, *pos*, *gene=None*)
Bases: object

HGVSPosition – Represent partial HGVS tags that refer to a position without alleles

Parameters

- **ac** (*str*) – sequence accession

- **type** (*str*) – type of sequence and coordinate
- **pos** (*str*) – sequence position
- **gene** (*str*) – gene symbol (may be None)

ac

gene

pos

type

hgvs.location

Provides classes for dealing with the locations of HGVS variants

This module provides for Representing the location of variants in HGVS nomenclature, including:

- integers and integer intervals (e.g., NC_012345.6:g.3403243_3403248A>C)
- CDS positions and intervals (e.g., NM_01234.5:c.56+12_56+14delAC)
- CDS stop coordinates (e.g., NM_01234.5:c.*13A>C)

Classes:

- *SimplePosition* – a simple integer
- *BaseOffsetPosition* – a position with datum, base, and offset for c. and r. coordinates
- *AAPosition* – an amino acid position (with AA)
- *Interval* – an interval of Positions

class hgvs.location.**AAPosition**(*base=None*, *aa=None*, *uncertain=False*)
Bases: object

aa

base

format(*conf=None*)

is_uncertain
return True if the position is marked uncertain or undefined

pos
return base, for backward compatibility

uncertain

validate()

class hgvs.location.**BaseOffsetInterval**(*start=None*, *end=None*, *uncertain=False*)
Bases: *hgvs.location.Interval*

BaseOffsetInterval isa Interval of BaseOffsetPositions. The only additional functionality over Interval is to ensure that the dutum of end and start are compatible.

check_datum()

class hgvs.location.**BaseOffsetPosition**(*base=None*, *offset=0*, *datum=<Datum.SEQ_START: 1>*, *uncertain=False*)
Bases: object

Class for dealing with CDS coordinates in transcript variants.

This class models CDS positions using a *base* coordinate, which is measured relative to a specified *datum* (CDS_START or CDS_END), and an *offset*, which is 0 for exonic positions and non-zero for intronic positions. **Positions and offsets are 1-based**, with no 0, per the HGVS recommendations. (If you"re using this with UTA, be aware that UTA uses interbase coordinates.)

hgvs	datum	base	offset	meaning
r.55	SEQ_START	55	0	RNA position 55
c.55	CDS_START	55	0	CDS position 55
c.55	CDS_START	55	0	CDS position 55
c.55+1	CDS_START	55	1	intronic variant +1 from boundary
c.-55	CDS_START	-55	0	5' UTR variant, 55 nt upstream of ATG
c.1	CDS_START	1	0	start codon
c.1234	CDS_START	1234	0	stop codon (assuming CDS length is 1233)
c.*1	CDS_END	0	1	STOP + 1
c.*55	CDS_END	0	55	3' UTR variant, 55 nt after STOP

base

datum

format(*conf*)

is_intronic
returns True if the variant is intronic (if the offset is None or non-zero)

is_uncertain
return True if the position is marked uncertain or undefined

offset

uncertain

validate()

class hgvs.location.**Interval**(*start=None*, *end=None*, *uncertain=False*)
Bases: object

end

format(*conf=None*)

is_uncertain
return True if the position is marked uncertain or undefined

start

uncertain

validate()

class hgvs.location.**SimplePosition**(*base=None*, *uncertain=False*)
Bases: object

base

format(*conf*)

is_uncertain
return True if the position is marked uncertain or undefined

uncertain

validate()

hgvs.posedit

implements a (position,edit) tuple that represents a localized sequence change

class hgvs.posedit.**PosEdit**(*pos=None*, *edit=None*, *uncertain=False*)
Bases: object

represents a **simple** variant, consisting of a single position and edit pair

edit

format(*conf=None*)
Formatting the string of PosEdit

length_change(*on_error_raise=True*)
Returns the net length change for this posedit.

The method for computing the net length change depends on the type of variant (dup, del, ins, etc). The length_change method hides this complexity from callers.

Parameters

- **self** (hgvs.posedit.PosEdit) – a PosEdit instance
- **on_error_raise** (*bool*) – whether to raise an exception on errors

Returns A signed int for the net change in length. Negative values imply net deletions, 0 implies a balanced insertion and deletion (e.g., SNV), and positive values imply a net insertion.

Raises **HGVSUnsupportedOperationError** – When determining the length for this variant type is ill-defined or unsupported.

There are many circumstances in which the net length change cannot be determined, is ill-defined, or is unsupported. In these cases, the result depends on the value of *on_error_raise*: when *on_error_raise* is True, an exception is raised; when False, the exception is caught and *None* is returned. Callers might wish to pass *on_error_raise=False* in list comprehensions to avoid dealing with exceptions.

pos

uncertain

validate()

hgvs.sequencevariant

represents simple sequence-based variants

class hgvs.sequencevariant.**SequenceVariant**(*ac*, *type*, *posedit*, *gene=None*)
Bases: object

represents a basic HGVS variant. The only requirement is that each component can be stringified; for example, passing pos as either a string or an hgvs.location.CDSInterval (for example) are both intended uses

ac

fill_ref(*hdp*)

format(*conf=None*)
Formatting the stringification of sequence variants

Parameters **conf** – a dict comprises formatting options. None is to use global settings.

See *hgvs.config*.

gene

posedit

type

validate()

Parsing and Formatting

hgvs.parser

Provides parser for HGVS strings and HGVS-related conceptual components, such as intronic-offset coordiates

class hgvs.parser.**Parser**(*grammar_fn=u'/home/docs/.cache/Python-Eggs/hgvs-1.3.0b2.dev1+g4d16efb-py2.7.egg-tmp/hgvs/_data/hgvs.pymeta'*, *expose_all_rules=False*)

Bases: object

Provides comprehensive parsing of HGVS varaint strings (*i.e.*, variants represented according to the Human Genome Variation Society recommendations) into Python representations. The class wraps a Parsing Expression Grammar, exposing rules of that grammar as methods (prefixed with *parse_*) that parse an input string according to the rule. The class exposes all rules, so that it's possible to parse both full variant representations as well as components, like so:

```
>>> hp = Parser()
>>> v = hp.parse_hgvs_variant("NM_01234.5:c.22+1A>T")
>>> v
SequenceVariant(ac=NM_01234.5, type=c, posedit=22+1A>T, gene=None)
>>> v.posedit.pos
BaseOffsetInterval(start=22+1, end=22+1, uncertain=False)
>>> i = hp.parse_c_interval("22+1")
>>> i
BaseOffsetInterval(start=22+1, end=22+1, uncertain=False)
```

The *parse_hgvs_variant* and *parse_c_interval* methods correspond to the *hgvs_variant* and *c_interval rules* in the grammar, respectively.

As a convenience, the Parser provides the *parse* method as a shorthand for *parse_hgvs_variant*: >>> v = hp.parse("NM_01234.5:c.22+1A>T") >>> v SequenceVariant(ac=NM_01234.5, type=c, posedit=22+1A>T, gene=None)

Because the methods are generated on-the-fly and depend on the grammar that is loaded at runtime, a full list of methods is not available in the documentation. However, the list of rules/methods is available via the *rules* instance variable.

A few notable methods are listed below:

parse_hgvs_variant() parses any valid HGVS string supported by the grammar.

```
>>> hp.parse_hgvs_variant("NM_01234.5:c.22+1A>T")
SequenceVariant(ac=NM_01234.5, type=c, posedit=22+1A>T, gene=None)
>>> hp.parse_hgvs_variant("NP_012345.6:p.Ala22Trp")
SequenceVariant(ac=NP_012345.6, type=p, posedit=Ala22Trp, gene=None)
```

The *hgvs_variant* rule iteratively attempts parsing using the major classes of HGVS variants. For slight improvements in efficiency, those rules may be invoked directly:

```
>>> hp.parse_p_variant("NP_012345.6:p.Ala22Trp")
SequenceVariant(ac=NP_012345.6, type=p, posedit=Ala22Trp, gene=None)
```

Similarly, components of the underlying structure may be parsed directly as well:

```
>>> hp.parse_c_posedit("22+1A>T")
PosEdit(pos=22+1, edit=A>T, uncertain=False)
>>> hp.parse_c_interval("22+1")
BaseOffsetInterval(start=22+1, end=22+1, uncertain=False)
```

parse(*v*)
parse HGVS variant *v*, returning a SequenceVariant

Parameters **v** (*str*) – an HGVS-formatted variant as a string

Return type *SequenceVariant*

Mapping

hgvs.assemblymapper

class hgvs.assemblymapper.**AssemblyMapper**(*hdp*, *assembly_name=u'GRCh38'*, *alt_aln_method=u'splign'*, *normalize=True*, *prevalidation_level=u'EXTRINSIC'*, *in_par_assume=u'X'*, *replace_reference=True*, *add_gene_symbol=False*, **args*, ***kwargs*)

Bases: *hgvs.variantmapper.VariantMapper*

Provides simplified variant mapping for a single assembly and transcript-reference alignment method.

AssemblyMapper is instantiated with an assembly name and alt_aln_method. These enable the following conveniences over VariantMapper:

- The assembly and alignment method are used to automatically select an appropriate chromosomal reference sequence when mapping from a transcript to a genome (i.e., c_to_g(...) and n_to_g(...)).
- A new method, relevant_trancripts(g_variant), returns a list of transcript accessions available for the specified variant. These accessions are candidates mapping from genomic to trancript coordinates (i.e., g_to_c(...) and g_to_n(...)).

Note: AssemblyMapper supports only chromosomal references (e.g., NC_000006.11). It does not support contigs or other genomic sequences (e.g., NT_167249.1).

Parameters

- **hdp** (*object*) – instance of hgvs.dataprovider subclass
- **replace_reference** (*bool*) – replace reference (entails additional network access)
- **assembly_name** (*str*) – name of assembly ("GRCh38.p5")
- **alt_aln_method** (*str*) – genome-transcript alignment method ("splign", "blat", "genewise")
- **normalize** (*bool*) – normalize variants
- **prevalidation_level** (*str*) – None or Intrinsic or Extrinsic validation before mapping

- **in_par_assume** (*str*) – during x_to_g, assume this chromosome name if alignment is ambiguous

Raises **HGVSError subclasses** – for a variety of mapping and data lookup failures

c_to_g(*var_c*)

c_to_n(*var_c*)

c_to_p(*var_c*)

g_to_c(*var_g*, *tx_ac*)

g_to_n(*var_g*, *tx_ac*)

g_to_t(*var_g*, *tx_ac*)

n_to_c(*var_n*)

n_to_g(*var_n*)

relevant_transcripts(*var_g*)
return list of transcripts accessions (strings) for given variant, selected by genomic overlap

t_to_g(*var_t*)

t_to_p(*var_t*)
Return a protein variant, or "non-coding" for non-coding variant types

CAUTION: Unlike other x_to_y methods that always return SequenceVariant instances, this method returns a string when the variant type is `n`. This is intended as a convenience, particularly when looping over `relevant_transcripts`, projecting with `g_to_t`, then desiring a protein representation for coding transcripts.

hgvs.variantmapper

Provides VariantMapper and AssemblyMapper to project variants between sequences using AlignmentMapper.

class hgvs.variantmapper.**VariantMapper**(*hdp*, *replace_reference=True*, *prevalidation_level=u'EXTRINSIC'*, *add_gene_symbol=False*)

Bases: `object`

Maps SequenceVariant objects between g., n., r., c., and p. representations.

g{c,n,r} projections are similar in that c, n, and r variants may use intronic coordinates. There are two essential differences that distinguish the three types:

- Sequence start: In n and r variants, position 1 is the sequence start; in c variants, 1 is the transcription start site.
- Alphabet: In n and c variants, sequences are DNA; in r. variants, sequences are RNA.

This differences are summarized in this diagram:

```
g ----acgtatgcac--gtctagacgt----      ----acgtatgcac--gtctagacgt----      ----
↪acgtatgcac--gtctagacgt----
      \          \/          /              \          \/          /              \  ⌋
↪       \/          /
c       acgtATGCACGTCTAGacgt          n       acgtatgcacgtctagacgt          r      ⌋
↪acguaugcacgucuagacgu
           1                                 1                                  1
p          MetHisValTer
```

The g excerpt and exon structures are identical. The gn transformation, which is the most basic, accounts for the offset of the aligned sequences (shown with "1") and the exon structure. The gc transformation is akin to gn transformation, but requires an addition offset to account for the translation start site (c.1). The CDS in uppercase. The gc transformation is akin to gn transformation with a change of alphabet.

Therefore, this this code uses gn as the core transformation between genomic and c, n, and r variants: All cg and rg transformations use ng after accounting for the above differences. For example, c_to_g accounts for the transcription start site offset, then calls n_to_g.

All methods require and return objects of type *hgvs.sequencevariant.SequenceVariant*.

Parameters

- **replace_reference** (*bool*) – replace reference (entails additional network access)
- **prevalidation_level** (*str*) – None or Intrinsic or Extrinsic validation before mapping

c_to_g(*var_c*, *alt_ac*, *alt_aln_method=u'splign'*)

Given a parsed c. variant, return a g. variant on the specified transcript using the specified alignment method (default is "splign" from NCBI).

Parameters

- **var_c** (hgvs.sequencevariant.SequenceVariant) – a variant object
- **alt_ac** (*str*) – a reference sequence accession (e.g., NC_000001.11)
- **alt_aln_method** (*str*) – the alignment method; valid values depend on data source

Returns variant object (*hgvs.sequencevariant.SequenceVariant*)

Raises **HGVSInvalidVariantError** – if var_c is not of type "c"

c_to_n(*var_c*)

Given a parsed c. variant, return a n. variant on the specified transcript using the specified alignment method (default is "transcript" indicating a self alignment).

Parameters **var_c** (hgvs.sequencevariant.SequenceVariant) – a variant object

Returns variant object (*hgvs.sequencevariant.SequenceVariant*)

Raises **HGVSInvalidVariantError** – if var_c is not of type "c"

c_to_p(*var_c*, *pro_ac=None*)

Converts a c. SequenceVariant to a p. SequenceVariant on the specified protein accession Author: Rudy Rico

Parameters

- **var_c** (SequenceVariant) – hgvsc tag
- **pro_ac** (*str*) – protein accession

Return type *hgvs.sequencevariant.SequenceVariant*

g_to_c(*var_g*, *tx_ac*, *alt_aln_method=u'splign'*)

Given a parsed g. variant, return a c. variant on the specified transcript using the specified alignment method (default is "splign" from NCBI).

Parameters

- **var_g** (hgvs.sequencevariant.SequenceVariant) – a variant object
- **tx_ac** (*str*) – a transcript accession (e.g., NM_012345.6 or ENST012345678)
- **alt_aln_method** (*str*) – the alignment method; valid values depend on data source

Returns variant object (*hgvs.sequencevariant.SequenceVariant*) using CDS coordinates

Raises **HGVSInvalidVariantError** – if var_g is not of type "g"

g_to_n(*var_g*, *tx_ac*, *alt_aln_method=u'splign'*)

Given a parsed g. variant, return a n. variant on the specified transcript using the specified alignment method (default is "splign" from NCBI).

Parameters

- **var_g** (hgvs.sequencevariant.SequenceVariant) – a variant object
- **tx_ac** (*str*) – a transcript accession (e.g., NM_012345.6 or ENST012345678)
- **alt_aln_method** (*str*) – the alignment method; valid values depend on data source

Returns variant object (*hgvs.sequencevariant.SequenceVariant*) using transcript (n.) coordinates

Raises **HGVSInvalidVariantError** – if var_g is not of type "g"

g_to_t(*var_g*, *tx_ac*, *alt_aln_method=u'splign'*)

n_to_c(*var_n*)

Given a parsed n. variant, return a c. variant on the specified transcript using the specified alignment method (default is "transcript" indicating a self alignment).

Parameters **var_n** (hgvs.sequencevariant.SequenceVariant) – a variant object

Returns variant object (*hgvs.sequencevariant.SequenceVariant*)

Raises **HGVSInvalidVariantError** – if var_n is not of type "n"

n_to_g(*var_n*, *alt_ac*, *alt_aln_method=u'splign'*)

Given a parsed n. variant, return a g. variant on the specified transcript using the specified alignment method (default is "splign" from NCBI).

Parameters

- **var_n** (hgvs.sequencevariant.SequenceVariant) – a variant object
- **alt_ac** (*str*) – a reference sequence accession (e.g., NC_000001.11)
- **alt_aln_method** (*str*) – the alignment method; valid values depend on data source

Returns variant object (*hgvs.sequencevariant.SequenceVariant*)

Raises **HGVSInvalidVariantError** – if var_n is not of type "n"

t_to_g(*var_t*, *alt_ac*, *alt_aln_method=u'splign'*)

hgvs.projector

Utility class that projects variants from one transcript to another via a common reference sequence.

class hgvs.projector.**Projector**(*hdp*, *alt_ac*, *src_ac*, *dst_ac*, *src_alt_aln_method=u'splign'*, *dst_alt_aln_method=u'splign'*)

Bases: object

The Projector class implements liftover between two transcripts via a common reference sequence.

Parameters

- **hdp** – HGVS Data Provider Interface-compliant instance (see *hgvs.dataproviders.interface.Interface*)

- **ref** – string representing the common reference assembly (e.g., GRCh37.p10)
- **src_ac** – string representing the source transcript accession (e.g., NM_000551.2)
- **dst_ac** – string representing the destination transcript accession (e.g., NM_000551.3)
- **src_alt_aln_method** – string representing the source transcript alignment method
- **dst_alt_aln_method** – string representing the destination transcript alignment method

This class assumes (and verifies) that the transcripts are on the same strand. This assumption obviates some work in flipping sequence variants twice unnecessarily.

project_interval_backward(*c_interval*)
project c_interval on the destination transcript to the source transcript

Parameters **c_interval** – an `hgvs.interval.Interval` object on the destination transcript

Returns c_interval: an `hgvs.interval.Interval` object on the source transcript

project_interval_forward(*c_interval*)
project c_interval on the source transcript to the destination transcript

Parameters **c_interval** – an `hgvs.interval.Interval` object on the source transcript

Returns c_interval: an `hgvs.interval.Interval` object on the destination transcript

project_variant_backward(*c_variant*)
project c_variant on the source transcript onto the destination transcript

Parameters **c_variant** – an *hgvs.sequencevariant.SequenceVariant* object on the source transcript

Returns c_variant: an *hgvs.sequencevariant.SequenceVariant* object on the destination transcript

project_variant_forward(*c_variant*)
project c_variant on the source transcript onto the destination transcript

Parameters **c_variant** – an *hgvs.sequencevariant.SequenceVariant* object on the source transcript

Returns c_variant: an *hgvs.sequencevariant.SequenceVariant* object on the destination transcript

hgvs.alignmentmapper

Mapping positions between pairs of sequence alignments

The AlignmentMapper class is at the heart of mapping between aligned sequences.

class hgvs.alignmentmapper.**AlignmentMapper**(*hdp*, *tx_ac*, *alt_ac*, *alt_aln_method*)
Bases: `object`

Provides coordinate (not variant) mapping operations between genomic (g), non-coding (n) and cds (c) coordinates according to a CIGAR.

Parameters

- **hdp** – HGVS Data Provider Interface-compliant instance (see *hgvs.dataproviders.interface.Interface*)

- **tx_ac** (*str*) – string representing transcript accession (e.g., NM_000551.2)
- **alt_ac** (*str*) – string representing the reference sequence accession (e.g., NC_000019.10)
- **alt_aln_method** (*str*) – string representing the alignment method; valid values depend on data source

alt_ac

alt_aln_method

c_to_g(*c_interval*)
convert a transcript CDS (c.) interval to a genomic (g.) interval

c_to_n(*c_interval*)
convert a transcript CDS (c.) interval to a transcript cDNA (n.) interval

cds_end_i

cds_start_i

cigar

cigar_op

g_to_c(*g_interval*)
convert a genomic (g.) interval to a transcript CDS (c.) interval

g_to_n(*g_interval*)
convert a genomic (g.) interval to a transcript cDNA (n.) interval

gc_offset

is_coding_transcript

n_to_c(*n_interval*)
convert a transcript cDNA (n.) interval to a transcript CDS (c.) interval

n_to_g(*n_interval*)
convert a transcript (n.) interval to a genomic (g.) interval

ref_pos

strand

tgt_len

tgt_pos

tx_ac

Validation and Normalization

hgvs.validator

implements validation of hgvs variants

class hgvs.validator.**ExtrinsicValidator**(*hdp*, *strict=True*)
Attempts to determine if the HGVS name validates against external data sources

validate(*var*, *strict=None*)

class hgvs.validator.**IntrinsicValidator**(*strict=True*)
Bases: object

Attempts to determine if the HGVS name is internally consistent

validate(*var*, *strict=None*)

class hgvs.validator.**Validator**(*hdp*, *strict=True*)
Bases: object

invoke intrinsic and extrinsic validation

validate(*var*, *strict=None*)

hgvs.normalizer

hgvs.normalizer

class hgvs.normalizer.**Normalizer**(*hdp*, *cross_boundaries=False*, *shuffle_direction=3*, *alt_aln_method=u'splign'*, *validate=True*)
Bases: object

Perform variant normalization

Initialize and configure the normalizer

Parameters

- **hdp** – HGVS Data Provider Interface-compliant instance (see *hgvs.dataproviders.interface.Interface*)
- **cross_boundaries** – whether allow the shuffling to cross the exon-intron boundary
- **shuffle_direction** – shuffling direction
- **alt_aln_method** – sequence alignment method (e.g., splign, blat)
- **validate** – whether validating the input variant before normalizing

normalize(*var*)
Perform sequence variants normalization for single variant

External Data Providers

hgvs.dataproviders.interface

Defines the abstract data provider interface

class hgvs.dataproviders.interface.**Interface**(*mode=None*, *cache=None*)
Bases: object

Variant mapping and validation requires access to external data, specifically exon structures, transcript alignments, and protein accessions. In order to isolate the hgvs package from the myriad choices and tradeoffs, these data are provided through an implementation of the (abstract) HGVS Data Provider Interface.

As of June 2014, the only available data provider implementation uses the Universal Transcript Archive (UTA), a sister project that provides access to transcripts and genome-transcript alignments. Invitae provides a public UTA database instance that is used by default; see the UTA page for instructions on installing your own PostgreSQL or SQLite version. In the future, other implementations may be availablefor other data sources.

Pure virtural class for the HGVS Data Provider Interface. Every data provider implementation should be a subclass (possibly indirect) of this class.

Parameters

- **mode** (*str*) – cache mode (None[default lru cache], 'learn', 'run', 'verify')
- **cache** (*str*) – local cache file name

data_version()

get_acs_for_protein_seq(*seq*)

get_assembly_map(*assembly_name*)

get_gene_info(*gene*)

get_pro_ac_for_tx_ac(*tx_ac*)

get_seq(*ac*, *start_i=None*, *end_i=None*)

get_similar_transcripts(*tx_ac*)

get_tx_exons(*tx_ac*, *alt_ac*, *alt_aln_method*)

get_tx_for_gene(*gene*)

get_tx_for_region(*alt_ac*, *alt_aln_method*, *start_i*, *end_i*)

get_tx_identity_info(*tx_ac*)

get_tx_info(*tx_ac*, *alt_ac*, *alt_aln_method*)

get_tx_mapping_options(*tx_ac*)

interface_version()

required_version = None

schema_version()

hgvs.dataproviders.uta

implements an hgvs data provider interface using UTA (https://github.com/biocommons/uta)

class hgvs.dataproviders.uta.**ParseResult**
Bases: urlparse.ParseResult

Subclass of url.ParseResult that adds database and schema methods, and provides stringification.

database

schema

class hgvs.dataproviders.uta.**UTABase**(*url*, *mode=None*, *cache=None*)
Bases: *hgvs.dataproviders.interface.Interface*

data_version()

get_acs_for_protein_seq(*seq*)
returns a list of protein accessions for a given sequence. The list is guaranteed to contain at least one element with the MD5-based accession (MD5_01234abc...def56789) at the end of the list.

get_assembly_map(*assembly_name*)
return a list of accessions for the specified assembly name (e.g., GRCh38.p5)

get_gene_info(*gene*)
returns basic information about the gene.

Parameters **gene** (*str*) – HGNC gene name

database results hgnc | ATM maploc | 11q22-q23 descr | ataxia telangiectasia mutated summary | The protein encoded by this gene belongs to the PI3/PI4-kinase family. This... aliases | AT1,ATA,ATC,ATD,ATE,ATDC,TEL1,TELO1 added | 2014-02-04 21:39:32.57125

get_pro_ac_for_tx_ac(*tx_ac*)
Return the (single) associated protein accession for a given transcript accession, or None if not found.

get_seq(*ac*, *start_i=None*, *end_i=None*)

get_similar_transcripts(*tx_ac*)
Return a list of transcripts that are similar to the given transcript, with relevant similarity criteria.

>> sim_tx = hdp.get_similar_transcripts('NM_001285829.1') >> dict(sim_tx[0]) { 'cds_eq': False, 'cds_es_fp_eq': False, 'es_fp_eq': True, 'tx_ac1': 'NM_001285829.1', 'tx_ac2': 'ENST00000498907' }

where:

- cds_eq means that the CDS sequences are identical
- es_fp_eq means that the full exon structures are identical (i.e., incl. UTR)
- cds_es_fp_eq means that the cds-clipped portions of the exon structures are identical (i.e., ecluding. UTR)
- Hint: "es" = "exon set", "fp" = "fingerprint", "eq" = "equal"

"exon structure" refers to the start and end coordinates on a specified reference sequence. Thus, having the same exon structure means that the transcripts are defined on the same reference sequence and have the same exon spans on that sequence.

get_tx_exons(*tx_ac*, *alt_ac*, *alt_aln_method*)
return transcript exon info for supplied accession (tx_ac, alt_ac, alt_aln_method), or None if not found

Parameters

- **tx_ac** (*str*) – transcript accession with version (e.g., 'NM_000051.3')
- **alt_ac** (*str*) – specific genomic sequence (e.g., NC_000011.4)
- **alt_aln_method** (*str*) – sequence alignment method (e.g., splign, blat)

tx_exons = db.get_tx_exons('NM_199425.2', 'NC_000020.10', 'splign') # len(tx_exons) 3

tx_exons have the following attributes:

```
{
    'tes_exon_set_id' : 98390
    'aes_exon_set_id' : 298679
    'tx_ac'           : 'NM_199425.2'
    'alt_ac'          : 'NC_000020.10'
    'alt_strand'      : -1
    'alt_aln_method'  : 'splign'
    'ord'             : 2
    'tx_exon_id'      : 936834
    'alt_exon_id'     : 2999028
    'tx_start_i'      : 786
    'tx_end_i'        : 1196
    'alt_start_i'     : 25059178
    'alt_end_i'       : 25059588
    'cigar'           : '410='
}
```

For example:

tx_exons[0]['tx_ac'] 'NM_199425.2'

get_tx_for_gene(*gene*)
return transcript info records for supplied gene, in order of decreasing length

Parameters **gene** (*str*) – HGNC gene name

get_tx_for_region(*alt_ac*, *alt_aln_method*, *start_i*, *end_i*)
return transcripts that overlap given region

Parameters

- **alt_ac** (*str*) – reference sequence (e.g., NC_000007.13)
- **alt_aln_method** (*str*) – alignment method (e.g., splign)
- **start_i** (*int*) – 5' bound of region
- **end_i** (*int*) – 3' bound of region

get_tx_identity_info(*tx_ac*)
returns features associated with a single transcript.

Parameters **tx_ac** (*str*) – transcript accession with version (e.g., 'NM_199425.2')

database output -[RECORD 1]–+———————- tx_ac | NM_199425.2 alt_ac | NM_199425.2 alt_aln_method | transcript cds_start_i | 283 cds_end_i | 1003 lengths | {707,79,410} hgnc | VSX1

get_tx_info(*tx_ac*, *alt_ac*, *alt_aln_method*)
return a single transcript info for supplied accession (tx_ac, alt_ac, alt_aln_method), or None if not found

Parameters

- **tx_ac** (*str*) – transcript accession with version (e.g., 'NM_000051.3')
- **alt_ac** (*str*) – specific genomic sequence (e.g., NC_000011.4)
- **alt_aln_method** (*str*) – sequence alignment method (e.g., splign, blat)

database output -[RECORD 1]–+———— hgnc | ATM cds_start_i | 385 cds_end_i | 9556 tx_ac | NM_000051.3 alt_ac | AC_000143.1 alt_aln_method | splign

get_tx_mapping_options(*tx_ac*)
Return all transcript alignment sets for a given transcript accession (tx_ac); returns empty list if transcript does not exist. Use this method to discovery possible mapping options supported in the database

Parameters **tx_ac** (*str*) – transcript accession with version (e.g., 'NM_000051.3')

database output -[RECORD 1]–+———— hgnc | ATM cds_start_i | 385 cds_end_i | 9556 tx_ac | NM_000051.3 alt_ac | AC_000143.1 alt_aln_method | splign -[RECORD 2]–+———— hgnc | ATM cds_start_i | 385 cds_end_i | 9556 tx_ac | NM_000051.3 alt_ac | NC_000011.9 alt_aln_method | blat

required_version = u'1.1'

schema_version()

class hgvs.dataproviders.uta.**UTA_postgresql**(*url*, *pooling=False*, *application_name=None*, *mode=None*, *cache=None*)

Bases: *hgvs.dataproviders.uta.UTABase*

close()

hgvs.dataproviders.uta.**connect**(*db_url=None*, *pooling=False*, *application_name=None*, *mode=None*, *cache=None*)
Connect to a UTA database instance and return a UTA interface instance.

Parameters

- **db_url** (*string*) – URL for database connection
- **pooling** (*bool*) – whether to use connection pooling (postgresql only)
- **application_name** (*str*) – log application name in connection (useful for debugging; PostgreSQL only)

When called with an explicit db_url argument, that db_url is used for connecting.

When called without an explicit argument, the function default is determined by the environment variable UTA_DB_URL if it exists, or hgvs.datainterface.uta.public_db_url otherwise.

```
>>> hdp = connect()
>>> hdp.schema_version()
'1.1'
```

The format of the db_url is driver://user:pass@host/database/schema (the same as that used by SQLAlchemy). Examples:

A remote public postgresql database: postgresql://anonymous:anonymous@uta.biocommons.org/uta/uta_20170707'

A local postgresql database: postgresql://localhost/uta_dev/uta_20170707

For postgresql db_urls, pooling=True causes connect to use a psycopg2.pool.ThreadedConnectionPool.

1.7 Privacy Issues

This page provides details about how the hgvs package works, with a focus on privacy issues that users may have. The intent is to provide users with enough information to assess privacy concerns for their institutions.

1.7.1 What's not done

No biologically-relevant data are collected or aggregated from any use of the hgvs package for any purpose. Furthermore, variant manipulation is entirely local. Sequence variants are never sent over the network.

Someg *hgvs* operations require additional data. For example, mapping variants between a genomic reference and a transcript requires transcript-specific alignment information. Currently, fetching addition data requires a network connection.

(We are considering whether and how to provide fully self-contained installations and do not require network access, but such is not available at this time.)

1.7.2 Data Provider Queries

hgvs requires a lot of specialized addition data to validate, normalize, and map variants. *All* queries for data are consolidated into a data provider interface that consists of 11 queries. The method signatures, including input arguments, are shown below with a discussion about privacy consequences.

fetch_seq(ac, start_i, end_i) This method fetches reference sequence in the context of the variant and is required in order to validate, normalize, and replace variant reference sequences. By sending accession and coordinates, it reveals a specific region of interest (and therefore genes and possible clinical conditions).

The current implementation, which fetches transcripts and genomic sequences from UTA, NCBI, and Ensembl, is a measure until we complete a comprehensive sequence archive.

data_version(), schema_version() Queries for meta data about the data provider.

get_acs_for_protein_seq(seq), get_gene_info(gene), get_tx_exons(tx_ac, alt_ac, alt_aln_method), get_tx_for_gene(gene), ge For all of these queries, the inputs are combinations of transcript accession, reference accession, gene name. These are likely too broad to constitute serious privacy concerns.

1.7.3 Information about current connections

The following is an example of the kinds of information available about a current connection as collected by PostgreSQL.

datname	uta
username	anonymous
application_name	hgvs-shell/0.4.0rc2.dev20+n97ead5bf0fed.d20150831
client_addr	162.217.73.242
client_hostname	invitae.static.monkeybrains.net
client_port	38318
backend_start	2015-08-31 22:58:26.411654+00
query_start	2015-08-31 22:58:30.669956+00
state_change	2015-08-31 22:58:30.673533+00
waiting	f
state	idle
query	select * from tx_exon_aln_v where tx_ac='NM_170707.3' and alt_ac='NC_000001.10' and alt_aln_method='splign' order by alt_start_i

Several of these merit discussion.

application_name Upon connection using the UTA data provider, a string containing the name of the python script and hgvs version are passed to the postgresql server. The string typically looks like `hgvs-shell/0.4.0rc2.dev20+n97ead5bf0fed.d20150831`. Clients may override the application_name when calling connect().

client_addr and client_hostname The source IP and hostname are available for current connections. For most clients, this will mean identifying an institution but not specific computers or individuals.

query The current or most recently executed query is visible. When accessed through the data provider, this field is limited to *Data Provider Queries*.

1.7.4 Historical connection information

Although we do have historical logs for database connections, they provide only date, time, and database connection. Currently, we do not log queries, although we might choose to periodically log certain queries for performance monitoring.

1.8 Contributing

hgvs is intended to be a community project. Contributions of source code, test cases, and documentation are welcome!

This section describes development conventions and practices for the hgvs project. The intention is to help developers get up-to-speed quickly.

1.8.1 Highlights

- Development occurs in the default branch. (Release branches are named for the major-minor release, e.g., 0.4.x.)
- Versioning follows Semantic Versioning.
- The code version is determined solely by the hg tags. This version appears in the package name (e.g., `hgvs-0.4.4-py2.py3-none-any.whl`) and in the version returned by `hgvs.__version__`. Updating to a specific version (*e.g.,* `hg up -r 0.4.0`) will get you exactly that version.
- All significant development *must* have an associated issue in hgvs issues; *create an issue if necessary*. Other changes *may* have have an issue. Please develop using a bookmark or branch named for the issue, such as `44-normalization`.
- Pull requests should be narrowly focused around a bug or feature. Discrete commits with good log messages facilitate review. Consider collapsing/squashing commits with `hgvs rebase --collapse ...` to make the PR concise. Submit PRs against the default branch head (or close to it).
- Abide by current *code style*. Use `make reformat` to reformat all code with yapf prior to submitting a PR.
- Email the hgvs-discuss mailing list if you have questions.
- Test your code with `make test` before you submit a PR.
- Currently, only Python 2.7 is supported. Support for Python 3.5 is slated for the next release (#190).

1.8.2 A Quick Contribution Example

- Fork the project at https://github.com/biocommons/hgvs/

  You will be able to make changes there and then submit your contributions for inclusion into the biocommons repo.

- Clone the project locally with

```
$ hg clone https://github.com/<your_username>/hgvs
```

- Create a virtualenv (recommended)

```
$ mkvirtualenv hgvs
```

  There are other ways to make python virtual environment. How you do this isn't important, but using a virtual environment is good practice.

- Prepare your environment

```
$ make develop
```

  The Makefile in hgvs wraps functionality in setup.py, and also provides many useful rules. See *make* for more information.

- Make a branch (for significant changes)

  If you expect to change multiple files, please work in a branch. Please name the branch like *141-formatter-class*.

- Code, code, code!

  You probably want to test code with:

```
$ make test
```

See *Local UTA* and *make* for tips on accellerating testing.

- Reformat code

  This command will reformat the entire package in-place.:

```
$ make reformat
$ hg commit -m 'fixes #141: implements Formatter class'
```

  Be sure to commit changes afterward!

- Commit and push:

```
$ make test
$ hg push
```

- Submit a pull request at the hgvs package web site.

1.8.3 Using a local/alternative UTA instance

- Install UTA from a PostgreSQL as described at in the UTA README.
- Specify an alternate UTA instance.

The easiest way to use a UTA instance other than the default is by setting UTA_DB_URL. The format is `postgresql://<user>:<pass>@<host>/<db>/<schema>`. For example:

```
postgresql://anonymous:anonymous@uta.biocommons.org/uta/uta_20140210
```

explicitly selects the public database, and

```
postgresql://localhost/uta/uta_20140210
```

selects a local instance. Developers can test connectivity like this:

```
$ UTA_DB_URL=postgresql://localhost/uta/uta_20140210 make test-quick
```

See hgvs/dataproviders/uta.py for current UTA database URLs.

1.8.4 Get Cozy with make

The hgvs package includes a GNU Makefile that aids nearly all developer tasks. It subsumes much of the functionality in setup.py. While using the Makefile isn't required to develop, it is the official way to invoke tools, tests, and other development features. Type *make* for hgvs-specific help.

Some of the key targets are:

develop Prepare the directory for local development.

install Install hgvs (as with python setup.py install).

test Run the default test suite (~4 minutes).

test-quick Run the quick test suite (~35s) of most functionality.

clean, cleaner, cleanest Remove extraneous files, leaving a directory in various states of tidiness.

docs Make the sphinx docs in `docs/build/html/`.

1.8.5 Code Style

The package coding style is based roughly on PEP8, with the following changes:

```
column_limit = 120
spaces_before_comment = 4
split_before_named_assigns = True
```

These code conventions are uniformly enforce by yapf. The entire code base is periodically automatically reformatted for consistency.

Variables

The following code variable conventions are used for most of the *hgvs* code base. They should be considered aspirations rather than reality or policy. Understanding these conventions will help uses and developers understand the code.

Note: A note on variable suffixes If a particular variant type is expected, a suffix is often added to variable names. *e.g.,* `var_c` in a function argument list signifies that a SequenceVariant object with type='c' is expected.

---

hgvs* a string representing an HGVS variant name.

var* a `hgvs.variant.SequenceVariant` object

pos

posedit

hgvs_position

1.8.6 Release Process

hgvs uses a home-grown tool, *clogger*, to generate change logs. This section documents the process. (Clogger will be released at some point, but it is currently really only executable by Reece.)

clogger's primary goal is to propose a preliminary changelog based on commit messages between specified release tags. That `.clog` file is a simple format like this:

```
clog format: 1; -*-outline-*-
* 0.4.1 (2015-09-14)
Changes since 0.4.0 (2015-09-09).
** Bug Fixes
*** fixes #274, #275: initialize normalizer with same alt_aln_method as␣
↪AssemblyMapper [43e174d6f8af]
*** fixes #276: raise error when user attempts to map to/from c. with non-coding␣
↪transcript [3f7b659f4f02]
```

`.clog` files should be edited for readability during the release process and committed to the repo (in `hgvs/doc/changelog/`).

A Makefile in the same directory generates an `.rst` file from the `.clog`. This file must also be committed to the repo. This file becomes the release changelog.

Finally, releases are bundled by major.minor versions in a file like `0.4.rst` (no patch level). That file must be edited to include the new release and committed to the repo.

Specific Example – 0.4.3 release

The 0.4.x branch has two recent changes for the 0.4.3 release. Here's how the release was prepared:

```
hg up 0.4.x
hg tag 0.4.3cl

cd doc/changelog
make 0.4.3cl.clog
mv 0.4.3cl.clog 0.4.3.clog
#edit 0.4.3.clog for readability
make 0.4.3.rst
#edit 0.4.rst to add 0.4.3 to index
```

`cd ../..` (hgvs top-level), then `hg status` should now look like:

```
M doc/changelog/0.4.rst
A doc/changelog/0.4.3.clog
A doc/changelog/0.4.3.rst
```

Check your work. Type `make docs`, then view `build/sphinx/html/changelog/0.4.3.html`.

Now we're ready to finish up:

```
hg tag --remove 0.4.3cl
hg com -m 'added docs for 0.4.3 release'
hg tag 0.4.3
hg push
make upload # (builds distribution and uploads to pypi)
```

1.9 Getting Help

hgvs always works and has no bugs. Furthermore, its interface is so easy to use that a manual is unnecessary.

Just kidding.

While hgvs is well-tested and has been used by many groups for several years, bugs, unexpected behaviors, and usages questions occur. Fortunately, there's now a small community of people who can help.

If you need help, please read the following sources first. Then, if you've still got a question, post to one of them.

If you have questions about the Variation Nomenclature Recommendations, consider posting your questions to the HGVS Facebook page.

1.9.1 hgvs-discuss Mailing List/Group

For general questions, the best source of information is the hgvs-discuss Google Group (https://groups.google.com/forum/#!forum/hgvs-discuss). It is publicly visible, but posting requires joining in order to control spam. The mailing list is the preferred way to reach the *hgvs* package authors. (Please do NOT send email directly to authors.)

1.9.2 Gitter Channel

We have a new gitter community at https://gitter.im/biocommons/hgvs. There's not much use yet, but there's a chance that you could get real-time replies there.

1.9.3 Bug Reports

If you think you've got a bug, please report it! Here are a few tips to make it more likely that you get a useful reply:

- Use the command-line tool *hgvs-shell* that comes with *hgvs* to prepare your bug report. Using *hgvs-shell* makes it easier for you to report the bug and make it easier for developers to understand it.
- Take the time to prepare a minimal example that demonstrates the bug. You are unlikely to get a reply if you submit a report that includes your own wrappers and tooling.
- Include the bug demonstration as text. A screenshot of a bug report is not reproducible.
- Include the values of *hgvs.__version__* and *hgvs.hdp.url*, and whether you're using seqrepo. (i.e., whether you specified HGVS_SEQREPO_DIR)
- hgvs-shell in an upcoming release will provide much of the above information for you, as shown below. Please use it.
- Include an explanation of the result you expected and why.
- Report the bug using github, which requires an account. If you don't have an account (and don't want to create one), sending the same information to the mailing list is acceptable.

```
$ hgvs-shell

################################################################################
hgvs-shell -- interactive hgvs
hgvs version: 1.1.3.dev11+ne7b6a1c3ec7a
data provider url: postgresql://anonymous:anonymous@uta.biocommons.org/uta/uta_
↪20170117
schema_version: 1.1
data_version: uta_20170117
sequences source: remote (bioutils.seqfetcher)

The following variables are defined:
* hp -- hgvs parser
* hdp -- hgvs data provider
* vm -- VariantMapper
* am37 -- AssemblyMapper, GRCh37
* am38 -- AssemblyMapper, GRCh38
* hv -- hgvs Validator
* hn -- hgvs Normalizer

hgvs_g, hgvs_c, hgvs_p -- sample variants as hgvs strings
var_g, var_c, var_p -- sample variants, as parsed SequenceVariants

When submitting bug reports, include the version header shown above
and use these variables/variable names whenever possible.

In [1]:
```

1.10 Frequently Asked Questions

1.10.1 Alignments for my transcript are not available. What can I do?

The short answer is: not much.

In order to project a variant between genomic and transcript coordinates, *hgvs* needs a sequence alignment. Sequence alignments are obtained from the Universal Transcript Archive (UTA), a compendium of transcripts and their genome alignments from multiple sources. Data are loaded from snapshots; the loading process is currently semi-automated and run irregularly.

UTA loads only high-quality alignments exactly as provided by the data sources. If an alignment is not provided by a data source, or if it fails filters recommended by NCBI, it won't be in UTA (with a small number of exceptions). Importantly, NCBI provides alignment data only for current transcripts against current assemblies; historical data are not available.

So, there are two common reasons that an alignment may not exist in UTA:

- The transcript was obsoleted before UTA started in 2014, or existed only *between* UTA snapshots.
- The transcript does not have any high-quality alignments.

If an alignment for a particular transcript-reference sequence pair and for a particular alignment method are not available, an exception like the following will be raised:

```
HGVSDataNotAvailableError: No alignments for NM_000018.2 in GRCh37 using splign
```

Currently, there is no way for users to provide their own alignments.

For example, UTA contains ten alignments for NM_000314 family of transcripts for PTEN:

transcript	genome	method
NM_000314.4	AC_000142.1	splign
NM_000314.4	NC_000010.10	blat
NM_000314.4	NC_000010.10	splign
NM_000314.4	NC_018921.2	splign
NM_000314.4	NG_007466.2	splign
NM_000314.5	NC_000010.10	splign
NM_000314.6	NC_000010.10	blat
NM_000314.6	NC_000010.10	splign
NM_000314.6	NC_000010.11	splign
NM_000314.6	NW_013171807.1	splign

A variant can be projected between any of the transcript, genome, and method combinations, and no other combination.

1.10.2 Why do I get different results on the UCSC browser?

The UCSC Genome Browser uses alignments generated by BLAT, which gives different results than the official alignments generated by NCBI using splign. Although BLAT and splign typically agree, there are many small differences in ambiguous alignments and even some substantial differences in a small number of transcripts. In some cases, the differences might cause a variant to be interpreted as coding using a splign alignment and non-coding by a BLAT alignment, or vice versa. Furthermore, one typically doesn't know which alignment set was used when publishing a variant. (Yes, that's a hot mess.)

1.10.3 Why do I get different results with Mutalyzer?

Some transcript-genome alignments contain indels. *hgvs* is careful to account for these indel discrepancies when projecting variants. In contrast, Mutalyzer does not account for such discrepancies. Therefore, the Mutalyzer results will be incorrect when projecting or validating a variant that is downstream of the first indel. For details and other examples, see https://www.ncbi.nlm.nih.gov/pubmed/30129167.

1.11 Change Log

1.11.1 1.2 Series

Warning: Python 2.7 versions of hgvs are now deprecated and will become unsupported on April 1, 2019. See [Migrating-to-Python-3.6](https://github.com/biocommons/org/wiki/Migrating-to-Python-3.6).

1.2.5 (2019-02-01)

Changes since 1.2.4 (2018-09-28).

Special Attention

Python 2.7 versions of hgvs are now deprecated and will become unsupported on April 1, 2019. See Migrating-to-Python-3.6.

Bug Fixes

- Fixes #546: relevant transcripts should be wholly within transcript bounds [86412924353b]

Internal and Developer Changes

- fix testing bug that caused py3.5 env to be undefined on travis [5146b07]

1.2.4 (2018-09-28)

Changes since 1.2.3 (2018-09-04).

Bug Fixes

- Closes #525: fix c_to_p bug with insertion of in-phase Ter (credit: @ianfab) [d40a71a6548b]
- backported test for #525 to 1.2 branch [cb769e9591d5]

Other Changes

- added missing changelog files for 1.2.3 [5d2cf1c]

1.2.3 (2018-09-05)

Changes since 1.2.2 (2018-08-09).

Bug Fixes

- Fixes #474, fixes #492: correct for stop gain located at termination codon

1.2.2 (2018-07-23)

Changes since 1.2.1 (2018-07-21).

Bug Fixes

- Closes #501: Add c_to_p support for inversion (#502)

1.2.1 (2018-07-21)

Changes since 1.2.0 (2018-07-15).

Bug Fixes

- Fixes #499: recognize whole-gene dup (c.-i_*j) and assume does not affect protein sequence [dc48d5d]

1.2.0 (2018-07-14)

Changes since 1.1.3 (2018-07-01).

Special Attention

This release contains a significant improvement in the accuracy of projecting variants in the vicinity of genome-transcript alignment gaps. Previously, hgvs handled only a limited number of cases. The new AlignmentMapper now handles all cases identified for projecting substitution, insertion, and deletion variants in the context of substituion, insertion, and deletion alignment discrepancies. [Credit: Meng Wang]

Deprecations and Removals

AlignmentMapper replaces IntervalMapper and TranscriptMapper. The latter are now deprecated and will be removed in the next release.

Bug Fixes

- Fixes #497: Honor normalize switch in AssemblyMapper [61d363e]

New Features

- Closes #208: Rewrite coordination mapping to provide better support for projecting variants in the vicinity of transcript-alignment gaps.

1.11.2 1.1 Series

1.1.3 (2018-07-01)

Changes since 1.1.2 (2018-03-31).

Bug Fixes

- Fixes #490: raises a NotImplementedError when a coding sequence is not divisible by 3 (#491) [35d72a577df5]

Other Changes

- added protein translation to README (how did we not have c_to_p there?!) [e7b6a1c3ec7a]
- switch to psycopg2-binary [9c1ec59a93ec]

Internal and Developer Changes

- added misc/proj-at-disc/ [1bdf4c40c750]
- added jupyter to etc/develop.reqs [9bdef16c24a2]
- update venv rules in Makefile [296feb69b7c5]
- update pypi link to new pypi.org site [76beb3424615]

1.1.2 (2018-03-31)

Changes since 1.1.1 (2017-11-24).

Bug Fixes

- Fix #483: fix bug when normalizing at first and last base (#484)
- Fix #480: fix validation of AAPosition (#485)
- Fix #431: fix length_change for ident var (#486) [5e59104cc739]
- Fix #476: fix c_to_p for dup at the end of cds (#478)
- Fix #488: unpin attrs package version by refactoring reftranscriptdata and altseqbuilder to not use closures or attrs [0e1d9f137642]

Internal and Developer Changes

- drop py3.5 from tox testing (Python 3.5 devel not available on Ubuntu 17.10) [3d311f8d2b26]
- omit coverage for utility and external source files [23b8b55eee87]
- pin setuptools_scm to 1.11.1, the last version known to work with hgvs [51ad9ad4b07a]

1.1.1 (2017-11-24)

Changes since 1.1.0.post1 (2017-07-11).

Bug Fixes

- Fixes #453: Fix get_tgt_length for m. var in normalizer [f7ec2a7a5037]
- Fixes #459: fix bug when raising exception on fetch_seq failure [650a97c715dc]
- Fixes #466: Wrong mapping result of identity variant without ref given (#468)
- Fixes #464: Make start and end position independent when start and end are equal [682f730bcfdb]

Other Changes

- Fixes #473: Pin attrs <17.3.0 due to 'ValueError: Cell is empty' [0ac8cffd262d]

Internal and Developer Changes

- tox now tests 2.7, 3.5 and 3.6
- catch KeyError from SeqFetcher (API change in seqfetcher) [da25364c6607]
- Invitae Only: Adds a new ncbi dataprovider that has been modeled on the uta dataprovider. (#472) [63b9c4a334eb]

1.1.0 (2017-07-11)

Changes since 1.0.0.post3 (2017-04-11).

Special Attention

This is the first version of hgvs that supports Python 3 (yay!). Continuous integration tests are now performed against Python 2.7, 3.5, and 3.6. Please report any issues.

Bug Fixes

- Closed #445: Fix normalization erros at start and end of transcript [56ed82a62f57]
- Closed #444: Fix normalizing var near the end of transcript

New Features

- Closed #424, #430: make no-change sequence optional for parsing, and do not include sequence by default on formatting [25fcf7a96158]
- Closed #427: Ensure c. coordinate within CDS bound
- Closed #439: Add method to explicitly close database connections [9f796476ba22]

- Handle the *cds_{start,end} is None* case explicitly, since None is not comparable to ints in python 3+. [13de480978de]
- Merged Python 3 support [deb08ea1f6fa]. Big thanks to Lucas Wiman and Counsyl for contributing Python 3 support!

Other Changes

- explicitly set and test _conn in UTA_postgresql.__init__. [faf5f37b77cd] Avoids sporadic errors during runtime shutdown in Python 3 (presumably due to non-deterministic object destruction order)

Internal and Developer Changes

- Added AssemblyMapper._fetch_TranscriptMapper() convenience method [cd2f21f2f8b3]
- Closed #343: Migrate from nose to pytest+tox [@lucaswiman] [b2263aed8ca0]. hgvs is now tested with tox in Python 2.7 and 3.5 environments.
- Reactivate CI testing with travis (previously drone.io) [ef23089c2c06]. master is currently testing on all commits https://travis-ci.org/biocommons/hgvs.png?branch=master

1.11.3 1.0 Series

1.0.0 (2017-04-08)

Changes since 0.4.0 (2015-09-09).

This is a major release of the hgvs package that includes new features and behavior changes. Some client code may require minor modification. (Note: Previously, we had tentatively called this release 0.5.0. The magnitude of the changes and the desire to migrate to public API versioning led us release as 1.0.0.)

See *Installing hgvs* for installation instructions.

Major Features and Changes

This section highlights import behavior or interface changes relative to the 0.4.x series. Code modifications are likely for most of the features listed below.

EasyVariantMapper renamed to AssemblyMapper, now with GRCh38 and PAR support. EasyVariantMapper was renamed to AssemblyMapper to better reflect its role. AssemblyMapper defaults to GRCh38. Transcripts in pseudoautosomal regions have alignments to X and Y. Previously, EVM would refuse to guess which alignment to use and raise an exception. AssemblyMapper has a new argument, *in_par_assume*, which enables callers to prefer X or Y alignments.

VariantMapper validates variants before mapping. Several bug reports resulted from attempting to project invalid variants, such as as variants with insertions between non-adjacent nucleotides. These generated exceptions or unexpected results. Intrinsic validation is now peformed before mapping and normalization, and callers may wish to catch these.

Fully local installations – no network access required. *hgvs* requires access to transcripts and sequences for most operations. By default, hgvs will use public remote resources at runtime, which incurs significant latency and, in principle, presents a minor privacy concern. While UTA has always been available for local installation, the more significant delay was in sequence lookup. A new package, SeqRepo, provides a local sequence database that is

synchronized with UTA. When used together, these packages completely obviate network connectivity and improve speed by approximately 50x.

hgvs transitions to public API versioning conventions. By transitioning from major version 0 ("initial development") to 1 ("public API"), we are indicating that the API is expected to be stable. In practice, this change will mean that x.y.z versions will clearly distinguish bug fix releases (increment z), backward-compatible feature additions (increment y), and API incompatible changes (increment x). See Semantic Versioning for more information.

Changes in p. formatting to better conform to current varnomen recommendations. Inferred changes with unknown p. effects are now represented with *p.?* rather than *p.(?)* (#393). In addition, silent SNV mutations now include the amino acid, as in *NP_000050.2:p.Lys2597=* rather than *NP_000050.2:p.(=)* (#317). Both changes improve conformance with current varnome recommendations.

By default, do not show reference sequence in dels and dups. For example, *NM_000059.3:c.22_23delAAinsT* would be shown as *NM_000059.3:c.22_23delinsT*. Users may configure *max_ref_length* (default 0) order to change this behavior (#404).

BaseOffsetPosition datums now use enums, defined in hgvs.enums. For example, previous *hgvs.location.SEQ_START* must be replaced with *hgvs.enums.Datum.SEQ_START* (#396).

Unknown protein effect are now internally represented with 'var.posedit=None'. This case is printed as *NP_01234.5:p.?* (#412).

Deprecations and Removals

- #349: drop support for dupN [0dbe440]
- #360: HGVSValidationError removed; used HGVSInvalidVariantError instead.

Bug Fixes

- #284: validation fails for g variants [512b882]
- #292: Fix bug in validator when validating m. variants and add tests [12d9b48]
- #308: validation across CDS start and CDS end boundaries [ac066ee], [8c8db03]
- #346: ensure that alignment starts at transcript position 0 [3af24b3], [9f29c87]
- #381: fix bug attempting to normalize p. variants; add issue test (test_issues.py) [834bed9]
- #393: fix inconsistent representation of unknown p. effect when inferred by c_to_p [3f1ac48]
- #409: Fix bug in normalizer when normalizing ident variant that is written as delins [94607ecc30da]
- #415: Limit assembly mapper to NC accessions [6056fd4414df]

New Features

- #105: configurable formatting [c8b9fd1]
- #249: Move intrinsic validation to models
- #253: Add p. validation support [3d3b9da], [ba943ae]
- #256: rename EasyVariantMapper to AssemblyMapper to better indicate functionality [d6697d6]
- #258, #330, #342: ensure that start and end datums are compatible [247d8bf]
- #260, #322: added tests to verify that exceptions are raised when mapping invalid variants [ac37ae0]

- #270: add is_intronic method to BaseOffsetPosition [4e40866]
- #282: preserve position in "identity" variants (e.g., norm(c.123A>A) => c.123= rather than c.=) [cc523ed]
- #295: raise exception when validating deletion length for intronic variants [4575ed8]
- #309: make errors more informative when coordinate is outside sequence bounds [d667d8b], [f4cd048]
- #314: support parsing identity variants [9116c72]
- #315: Validate accession type pairs [be90d50]
- #316: provide generalized transcript-to-genome projects to handle coding and non-coding transcripts transparently [26006c7]
- #317: Output silent p. variants according to HGVS recommendations [4babbb5]
- #319: added PosEdit.length_change() method [c191567], [c71a48b], [c70fded]
- #326: provide special handling for disambiguating alignments in pseudoautosomal regions [acc560e]
- #330: datum-matching [e05674b], [461ccd7]
- #336: add hgvs-shell as excutable for exploration, debugging, bug submission [f6b6c3c]
- #356: add position comparision operators [4f7f7e4]
- #365: graded validation
- #366: move validation to variantmapper
- #372: rename hgvs/variant.py to hgvs/sequencevariant.py [2f69d65], [ad604fd]
- #379: move replace_reference to variantmapper (from evm) [c0f4be1]
- #386: reject discontiguous alignments [ea2527c]
- #391: Attempt reconnection if db connection is lost [2aef5fac3a61]
- #399: validators should raise only HGVSInvalidVariantError exceptions
- #404: Implement max_ref_length in formatter and don't show reference sequence by default

Other Changes

- #276: raise error when user attempts to map to/from c. with non-coding transcript [aaa0ff5]
- #363: update railroad diagram [3e23e10]

Internal and Developer Changes

- #236: migrate from built-in seqfetcher to bioutils seqfetcher [5e9a951]
- #237: Mock testing data; dropped unmaintained sqlite-based tests
- #287: ensure that modules are also getting doctested (continues #287) [3cbe93a]
- #347: Replace recordtype with attrs
- #395: get ThreadedConnectionPool sizes from config [a2a216c]
- #397: use hgvs.config for VariantMapper.__init__() [154cf5e]
- #400: make hdp cache mode (for testing) settable via HGVS_CACHE_MODE environment variable [09303c7]
- removed sqlite-based uta dataprovider; updated reqs in etc [e8c9d8d85d35]

1.11.4 0.4 Series

0.4.14 (2017-05-19)

Changes since 0.4.13 (2016-12-12).

New Features

- Closed #439: Add method to explicitly close database connections [BROKEN: 5a876fd2d1ec]

0.4.13 (2016-12-12)

Changes since 0.4.12 (2016-12-06).

Bug Fixes

- closes #390: fix missing HGVSError import in variantmapper [BROKEN: 9e3bee72a349]

0.4.12 (2016-12-06)

Changes since 0.4.11 (2016-09-15).

Bug Fixes

- #386: reject discontiguous alignments [BROKEN: 839a6fc36c7d]

Other Changes

- Minor typo corrections on quick_start.rst [BROKEN: 49bb4ac246f1] (PR #53 from kmcallenberg)

0.4.11 (2016-09-15)

Changes since 0.4.10 (2016-09-13).

Other Changes

- fixed #357: reenable parsing of sequence with inversion (backed out #340) [BROKEN: 881c58dda474]

0.4.10 (2016-08-16)

Changes since 0.4.9 (2016-08-01).

Bug Fixes

- fixes #336: add hgvs-shell as excutable for exploration, debugging, bug submission [BROKEN: 8ae7f072abc1]
- fixes #346: pushed alignment validation into dataprovider get_tx_exons() to cover use in normalizer [BROKEN: 0bc61059562c]

Other Changes

- closes #352: use https for seqfetcher [BROKEN: ed0655b1bb2b]

0.4.9 (2016-08-01)

Changes since 0.4.8 (2016-07-19).

Special Attention

A small number of alignments provided by NCBI do not begin at the transcript start. These exist in UTA as-is and lead to incorrect mapping and validation. Issue #346 contains the list of 52 transcripts in 37 genes which exhibit this issue; please review prior results. **hgvs will now refuse to use such alignments.**

Bug Fixes

- #346 (partial fix): ensure that alignment starts at transcript position 0 [BROKEN: ab402bf020c6]
- fixes #338: check position range limit when normalizing [BROKEN: da5f1fbcf76d]
- fixes #285, #334, #335, #324, #340: inversions parsing, formatting, and normalization [BROKEN: 29a7b8634b01]
- fixes #340: do not accept sequence following inv [BROKEN: f76e1cb83422]

0.4.8 (2016-07-19)

Changes since 0.4.7 (2016-06-27).

Bug Fixes

- fixes #337: soft-pin bioutils >=0.1.0,<0.2.0 [BROKEN: 13620e943e0c]

0.4.7 (2016-01-23)

Changes since 0.4.6 (2016-06-27).

Bug Fixes

- fixes #310: Fix wrong start position when normalizing some variants [BROKEN: 734c08f18ea1]. Thanks to Meng Wang.

0.4.6 (2016-06-27)

Changes since 0.4.5 (2016-04-01).

Bug Fixes

- fixes #308: fix issues with validating across CDS start and CDS end boundaries [BROKEN: ce6995941984]

Other Changes

- closes #309: make errors more informative when coordinate is outside sequence bounds [BROKEN: e6e0decdad8e]
- closes #295: raise error when attempting to validate del length in intronic variants [BROKEN: 13674d3c6d14]

Internal and Developer Changes

- fix issues with release docs for 0.4.x layout [BROKEN: 52b2358fed02]

0.4.5 (2016-03-31)

Changes since 0.4.4 (2015-12-15).

Special Attention

- The *_execute()* method of the UTA data provider was removed.

As part of addressing bug #321, this *internal* method was removed. Deprecation notices will not be issued for internal methods. (By Pyhon convention, tokens beginning with an underscore are considered private to the package or module.)

Bug Fixes

- fixes #321: use context manager to obtain and release cursors [BROKEN: 70c13e5a0643]

New Features

- closes #319: added PosEdit.length_change() method [BROKEN: fa5bb5fb9a50]

Other Changes

- closes #299: migrate 0.4.x branch docs to rtd theme [BROKEN: 3e016264457d]

0.4.4 (2015-12-15)

Changes since 0.4.3 (2015-12-06).

Bug Fixes

- fixes #282: preserve position in "identity" variants (e.g., norm(c.123A>A) => c.123= rather than c.=) [BROKEN: 5e6fd1524204]. (Reported by Stephan Pabinger.)
- fixes #294: extend variant type checks in validator [BROKEN: e28b5a525f6e]
- fixes #292: Fix bug in validator when validating m. variants and add tests [BROKEN: 64e31808a760]

Other Changes

- stopgap for #253: issue warning that p. validation is unsupported [BROKEN: a9bd9ab405bc] (Reported by Ram Srinivasan.)

0.4.3 (2015-12-04)

Changes since 0.4.2 (2015-09-30).

New Features

- closes #281: install hgvs-shell executable with package [BROKEN: bece4e961cd4]

Other Changes

- closes #289: work around pycharm bug PY-4213 [BROKEN: 19c0d4fefbfd]
- added 0.4.2 changelog (after the tagged commit :-() [BROKEN: 4a596322bceb]

0.4.2 (2015-09-30)

Changes since 0.4.1 (2015-09-14).

Bug Fixes

- fixes #284: validation fails for g variants [BROKEN: 9732eaf5be1c]

0.4.1 (2015-09-14)

Changes since 0.4.0 (2015-09-09).

Bug Fixes

- fixes #274, #275: initialize normalizer with same alt_aln_method as EasyVariantMapper [BROKEN: 43e174d6f8af]
- fixes #276: raise error when user attempts to map to/from c. with non-coding transcript [BROKEN: 3f7b659f4f02]

0.4.0 (2015-09-09)

Changes since 0.3.7 (2015-06-23). See issues at milestone 0.4.0.

Special Attention

- #227: x_to_r and r_to_x methods were renamed to x_to_n and n_to_x as part of support for non-coding transcripts.
- #231: The UTA data provider will use a recently updated database by default (uta_20150827). Clients with custom configurations should use postgresql://anonymous:anonymous@uta.biocommons.org/uta/uta_20150827. (Note the change of hostname, username, and password as well; see Deprecations.)
- #238: Most methods now raise HGVSDataNotAvailableError when expected data is not available. Previously, None was returned for some methods.
- #244: Removed cache_transcripts argument from VariantMapper. This argument was deprecated in 0.3.0 and is now obsolete. Dataproviders are now expected to cache data.
- #246: Remove hgvsX_to_hgvsY methods. These methods were deprecated in 0.3.0 and are now obsolete.
- #247: Dup and Repeat "seq" instance variable renamed to "ref" for consistency.
- EasyVariantMapper, Normalizer, and Validator now fetch sequence data at runtime, which may raise performance and privacy concerns. Users may wish to read Privacy Issues in the documentation.

Deprecations

- UTA: UTA now uses anonymous:anonymous as the username:password. uta_public:uta_public will be obsolete shortly.

Bug Fixes

- #248: Don't raise validation exception when del sequence is empty [BROKEN: b6c07d329d36]

New Features

- #44: Added variant normalization and use during mapping. Thanks to Meng Wang and Kevin Jacobs for contributions. (pull request #17)
- #168: EasyVariantMapper supports replacing the reference sequence during mapping and enabled by default.
- #227: Implement initial support for non-coding transcripts.
- #230: Allow full IUPAC for NA and AA, with tests. Previously, the grammar admitted only ACGTU.
- #233: Added get_similar_transcripts() to data UTA provider to expose UTA's tx_similarity view.
- #234, #241: Preferentially use transcript-protein accession associations from RefSeq when mapping c. to p. variants. Previously, when multiple protein accessions were associated with a single distinct sequence, the p. accession was arbitrary.
- #236, #240: Added seqfetcher.SeqFetcher to fetch sequences from NCBI & Ensembl
- #250: Implemented configuration module; hgvs.global_config is initialized once and available globally

- #251: Parens now optional around p. edits; default is enabled per HGVS spec (hgvs.global_config.mapping.inferred_p_is_uncertain)
- #255: variants normalized by EasyVariantMapper by default [BROKEN: 1b85d4deabc3]
- #261: Replace reference default from config (hgvs.global_config.mapping.replace_reference)
- UTA is now available as a docker image for local installation. See *Local Installation of UTA (optional)*.

Other Changes

- #213: Clarify warning message when validating intronic variants.
- #254: Support inversion, conversion, and nadupn variants
- Added misc/experimental/tx-seq-discrepancies to identify genomic locations of reference-transcript discrepancies
- Added variant context method to evm (temporary location, but useful for debugging)
- HGVSInternalError now subclasses HGVSError (not Exception) [BROKEN: ff6cd4dc51dc]
- Lots of documentation updates.
- Raise HGVSParserError (instead of ometa.runtime.ParseError) when parsing fails [BROKEN: efa93fe29d15]
- The UTA data provider now checks for the requested schema on connection and provides more informative errors on failure.
- hdp.data_version returns schema name for UTA since that that is the conventional use.
- Use autocommit to prevent transaction overhead and locks [BROKEN: 65d69e41716e]

Internal and Developer Changes

- #263: Trying out a new tag-based changelog mechanism for 0.4.0 and 0.4 series.
- All code will be mercilously reformmated with yapf occasionally using .style.yapf
- Build and upload wheel packages (in addition to existing eggs and tarballs)
- Docs significantly overhauled and moved to readthedocs.org with automatic webhook-based building
- Enable users to set application_name when connecting [BROKEN: 835ac7771909]
- _UTA_URL_KEY (dev use only) will switch URLs to any in hgvs/_data/defaults.ini
- on import of hgvs, emit logging info line w/version [BROKEN: aa97f2c1cdc8]
- sped up most tests by using setUpClass() rather than setUp() [BROKEN: a6d227f6a3e0]

This is the monolithic changelog for the 0.0, 0.1, 0.2, and 0.3 series of hgvs releases. Beginning with 0.4, changes will be recorded in release-specific files; see *Change Log*.

1.11.5 0.3 Series

0.3.7 (2015-06-23)

Client Changes

- #233: Expose UTA's notions of transcript similarity via the UTA data provider. See get_similar_transcripts().

- #236: Added seqfetcher.SeqFetcher to fetch sequences from NCBI & Ensembl
- #199: Improved installation documentation re: PostgreSQL dependency
- #232: Migrated to major.minor versions for schemas and schema provider-client compatibility; "compatible" := (provided x == required x) ^ (provided y >= required y)
- misc/experimental/vcf-add-hgvs: optionally generate coding variants
- numerous doc updates

Internal and Developer Changes

- add missing requests library to setup.py (only affected developers)
- updated bioutils version in setup.py

0.3.6 (2015-06-02)

- #228: IndexError when schema name is empty
- #228: updated CHANGELOG
- hgvs/edit.py: doc string indentation fix

0.3.5 (2015-05-19)

- #219: remove validation requirement that ref != alt
- #220: Do not modify cached results when building CIGAR (pkaleta)
- #226: support schema names in db urls; standardized search_path handling; merge connection pool and single-threaded client classes
- added AUTHORS

0.3.4 (unreleased)

0.3.3 (2014-08-28)

- #194: fix bug when reverse complementing nucleotides parsed from unicode
- #197: use utf-8 coding, unicode, and all py3k __future__ features in all source
- #198: documentation improvements
- #202: implement mutalyzer comparisons
- #203: return HGVSParseError instead of ometa.runtime.ParseError for parsing errors
- #205: fix "base" bias for the exact middle of an odd-length intron
- #206: make get_tx_for_region return only transcripts with alignment data
- added flake8 configuration
- added regression test framework (tests/data/gcp/regression.tsv)

0.3.2 (2014-07-12)

- #194: fix bug when reverse complementing nucleotides parsed from unicode

0.3.1 (2014-07-12)

- #193: fix lookup table for NC_000014.8 (was .10)
- #192: deprecated VariantMapper cache_transcripts param and replaced with always-on lru cache in uta data provider

0.3.0 (2014-06-19)

- #103: significantly updated documentation
- #162: provide simplified mapping interface, EasyVariantMapper
- #171: integrate the data provider interface into hgvs, obsoleting bdi. See hgvs.dataproviders.*
- #177: rename mapping functions to x_to_y (dropping "hgvs" prefix)
- #180: made set_uncertain an internal method (_set_uncertain)
- #181: renamed hgvs.hgvsmapper.HGVSMapper to hgvs.variantmapper.VariantMapper
- #184: rename HGVSPosition.seqref to ac
- #185: enable validator to use HDPI to fetch sequence data; mfdb now required only for genomic sequences
- Makefile: print machine info during testing to calibrate/debug timing probs
- moved hgvs/data to hgvs/_data to emphasize it is internal and avoid tab completion on it
- remove unused args from VariantMapper.c_to_p()
- replace u1/uta1 references with hdp; update docs
- replaced bdi with hdp when referencing the data provider; tests pass
- setup.py: removed nose-timer (appeared to cause problems with pip install)
- standardize exception names with "HGVS" prefix
- updated examples/manuscript-example; other minor changes

1.11.6 0.2 Series

0.2.2 (2014-06-12)

- #103: significantly updated documentation
- #142: added BIC test cases
- #167: disable the any_variant rule because it is confusing
- #179: added quick and extra tags to tests; updated Makefile to support make test, test-quick, test-extra; removed test_hgvs_parser_real (but kept gcp version)
- added support for testing models ("models" attr and test-models)

0.2.1 (2014-06-11)

- #157: don't reverse complement numeric "sequences" (as in del26)
- #159: Update comment in tests/data/ADRA2B-dbSNP.tsv
- #161: transform examples to sphinx doc (+upload)
- #167: disable the any_variant rule because it is confusing
- #175: added type to NADupN and Copy edit classes
- Added Important Notes section in README.rst
- Makefile: "test" target should depend on "setup" after all
- added example for stringification to README.rst
- added examples/Manuscript Example.ipynb
- added installation status (from hgvs-integration-test at travis-ci) and build status (from drone.io)
- hgvsmapper: use deepcopy when converting edits
- removed unused sphinx_pypi_upload.py
- updated examples to use uta1

0.2.0 (2014-03-09)

- updated README.rst example to use uta1; added .rst files to nosetest testing
- added ci-test-ve; switched to hgtools 5.0 use_vcs_version in setup.py
- take 1 on reconcililing test differences between internal jenkins and drone.io
- removed accidental tag (!); added sphinxcontrib-fulltoc to setup.py

1.11.7 0.1 Series

0.1.11 (2014-03-05)

- removed accidental tag (!); added sphinxcontrib-fulltoc to setup.py
- updated package metadata; removed requirements.txt; tests pass

0.1.9 (2014-03-05)

- #40: added additional tests
- #114: add test that checks that all rules have been tested - and add tests for rules that were missed!
- #135: add more tests; fixed and enabled tests previously commented out
- #147: update tests to use updated sqlite test DB
- Added U14680.1 (BIC tx) to grammar test
- ExtrinsicValidator should not guess about bdi and mfdb sources; instead require caller to specify
- Fixed an un-handled case for parsing AA frameshifts - short form, e.g. "Ala97fs" (no alt AA). Added tests.
- Makefile, setup,py, setup.cfg sync with sibling projects

- Merged hgvs_using_uta1 into default
- Merged in extrinsic_validation (pull request #5)
- Remove redundant test
- added Validator class that wraps instrinsic and extrinsic validation
- added bdi accession testing
- added codeship status badge to README.rst, for testing
- added creating-a-variant example
- added sbin/get-dbsnp-tests-for-gene
- added tests from dbSNP for 6 new gene; fixed probs with uncertainty and Terd+ in existing tests
- bug fixes for uta1 integration; all tests pass except for sqlite db test
- checking cigar ref tgt orientation
- cigar intron count fix
- cut DNAH11 tests to representative set (apx 80% cut)
- finished integrating uta1 into hgvs and started updating tests
- fixed DNAH11-dbSNP tests
- fixed bug when falling off transcripts
- hgvsmapper is updated with uta1 requirements. testing modifications using hgvs-shell
- removed accession test from extrinsic validator (sequence lookup covers accession lookup)
- removed codeship badge
- renamed ~Validation to ~Validator to keep with class-as-actor naming scheme
- starting external validation with bdi
- testing
- trivial change to tickle codeship build
- updated edit type and tests to include identity for sub e.g., T>T
- updated external validation using bdi; added identity edit type for sub T>T; added HGVSValidationException class; added sample tests for mfdb
- updated package metadata; removed requirements.txt; tests pass
- upped bdi min version to >=0.1.0 (interface1)
- use pip installation status as build status since that's what users will experience
- working through updating TM and IM. HM g_to_c appears to work

0.1.8 (2014-01-22)

- updated README.rst example for bdi connect()

0.1.7 (2014-01-22)

- #106, #108: parse uncertain hgvsp/hgvsr; converter produces uncertain hgvsp.
- #110, #111: handle cases of entire gene deletion (p.0?) and stop codon in frame (p.?). Updated tests.
- #65, #89: can now parse Met1? and ext*N; removed extra fs parsing from delins.
- #65: cleanup; AASub can go back to being a subclass of AARefAlt
- #65: def_p_pos needs to accept term13 as well as aa13 for ext; tests updated.
- #65: fixed an ordering bug; added tests.
- #65: fs/ext are now their own pro_edit types; they correspond to their own class objects. 5' extensions and 3' extensions can be parsed. Tests updated.
- #65: should be stringifying * as Ter; fixed code in 2 lines & tests in many.
- #65: tighten ext rules; require a number for new start positions.
- #90: added dup in hgvsmapper; allowed rev complement util to handle None (was triggering exceptions); added tests for dup.
- #91: add extension support for parsing copyN and DupN
- #91: make adding default totally extendable by allowing additional imports for the base grammar (default empty list)
- #91: simplest implementation of parsing copyN, dupN - added directly to grammar (no extension)
- #99: fix aa13t parsing
- #99: fix aa13t parsing, take 2; tests pass (including G* test)
- #99: re-enable tests related to this issue.
- Fixed a bug where del5insT was getting stringified as "5>T"
- added datum to range checking
- added datum to range checking
- added edit type as a property to the edit object; updated tests; added examples to hgvs-shell
- added edit type as a property to the edit object; updated tests; added examples to hgvs-shell
- close anonymous branch
- closed experimental dev branch
- closed hgvsvalidator feature branch on wrong default branch (grafted to default)
- doc updates and Makefile fix after fouled merge
- fixed minor doc typos
- hgvsc_to_hgvsp - ac defaults to None; seems better than forcing the user to pass 'None' as a param if they want the protein accession looked up.
- iv grammar branch
- make doc is broken & not used; removing it from make ci-test for now.
- merged in validator (pull request #4)
- minor change to rebase
- removed links section from README

- renamed hgvsvalidator to validator and corresponding test; corrected start-end check added tests
- revised intrinsic validator and tests; deleted requests from setup.py
- updated README.rst example for bdi connect()
- updated docs to point back to pythonhosted
- updated installation.rst
- updated ipython notebook examples
- updated railroad building
- updated railroad in docs
- updated the fragile railroad building again

0.1.6 (2014-01-11)

- updated docs to point back to pythonhosted
- added setuptools to requirements.txt
- updated requirements.txt
- fixed bug in setup.py re: classifiers

0.1.5 (2014-01-11)

- fixed bug in setup.py re: classifiers

0.1.4 (2014-01-11)

- #97: a bagillion doc updates; branch closed

0.1.3 (2014-01-11)

- #60: 1st stab at grammar tests from the bottom-up (through locations/definite positions). (See header in test_hgvs_grammar_full.py for details.) Also added a few error checking tests.
- #60: drop None from SequenceVariant (use case - only parsing an edit); grammar update for offset
- #60: implement cleanup; distributed remaining items to separate issues.
- #73: migrate hgvs to bdi-based protein accession lookup
- #90: fixed typo for delins and ins for parsing hgvsp
- #92: add a subclass of AARefAlt (AASub) which overrides __str__ to get the representation right; grammar update
- #92: fix error in NARefAlt
- #93: added *variant* liftover for HGVS projector, with tests
- #93: implemented HGVS projector for interval liftover
- #96: cleanup and test update
- #96: deleting tests/data

- #96: fix file
- #96: name cleanup
- #96: removed nightly test target
- #96: short set of real data for gcp parsing
- #97: a bagillion doc updates; branch closed
- #97: major doc restructuring, cleanup, additions
- A few more basic tests
- Add parser test which just tries to parse all the cvids (g, c and p) - currently skips unsupported forms. Also tweaked the r variants in the all cvid file (T should be U).
- Add some basic intervalmapper tests based on the coverage results
- Fill in more protein edit tests
- Fixed a bug breaking n_edit and m_edit; updated tests.
- Make documentation more Sphinx-friendly
- More grammar tests; simplified dup check for hgvsc to p conversion
- Tweak HGVSp expected so an edit creating a stop codon is represented by Ter instead of * (to match hgvs string code)
- add alternative UTA_DB_URL options to Makefile; cleanup eggs in cleanest (not cleaner) and bdist et al. in cleaner (not cleanest)
- added .travis.yml
- added a projector example
- added classifiers and keywords to setup.py
- added license to docs
- added railroad diagram to docs
- additional grammar tests - HGVS edits are failing commented out for now
- bug fix: make test was running nightly tests
- build reST doc for railroad grammar
- code cleanup
- commenting out test until I am in a place where I can run it
- doc updates
- eliminated most sphinx warnings
- lots of doc restructuring and consolidation
- minor cleanup
- more grammar tests
- removed reST examples
- sync default into branch
- sync default into dev
- updated README with pypi info

- updated installation
- updated misc/hgvs-shell for new bdi.uta0.connect()
- updated railroad diagram to include version number
- updated sphinx doc/source/conf.py
- yet more doc changes

0.1.2 (2014-01-05)

- #85: adapted hgvs to bdi with runtime-selectable UTA connections
- updated README with pypi info
- doc updates
- now depend on uta and bdi from PyPI (not dependency_links); sync'd Makefile and setup.py with uta; updated test and docs targets

0.1.1 (2014-01-03)

- #64: handle the following: (1) indel crosses stop codon; (2) indel crosses start codon; need to retest on full suite
- #64: update 4 tests to reflect p.Met1? behavior for deletions crossing from 5'utr to cds:
- #83: cleanup fs* cases where mutalyzer assigns fs*N where N = end of transcript instead of an actual stop codon (expected result is now fs*?)
- #83: comment out tests that need review/cleanup (and added comment); fixed tests where expected result was incorrect (still need to check tests w/ no expected result)
- #83: fill in intronic variants with expected hgvsp results (p.?) per curators
- #84: ext with no stop codons are represented as ext*? - updated tests accordingly
- #84: fix expected result
- Turn off dbg
- Turn off more dbg
- added *lots* of documentation
- added Apache license and code boilerplate to all source files and scripts
- doc updates
- fix coverage by calling tests via python setup.py nosetest; fix test name
- logo: rotated, moved to subdir, created favicon
- made png and ico logos transparent
- moved sphinx sources to doc/source and updated configs
- now depend on uta and bdi from PyPI (not dependency_links); sync'd Makefile and setup.py with uta; updated test and docs targets
- removed test-setup-coverage from Makefile dependencies (put in setup.py instead)
- s/locusdevelopment/invitae/
- updated doc static images

- updated hgvs-logo.png per Makefile
- updated setup.py "license" attribute
- vastly improved sphinx documentation. More to do

0.1.0 (2013-12-30)

- #52: generate syntax/railroad diagrams (in misc/railroad/)
- #56: updated tests; fixed fs*N (only one still broken)
- #62: synchronized setup files among UTA program components
- #66: added support for p.0, p.=, p.?, p.(=), p.(?), with tests
- #66: updated grammar for p.0, p.=, p.?, p.(=), p.(?) to reject invalid p.(0), etc.
- #72: update hgvs to use bdi (no direct connections to uta anymore)
- Close branch jenkins.
- Convert test input and consumer to use 4-column format
- Fix extension for frameshift case; update test to get around dupN (trim the N)
- Fix tag
- Last cleanup before merge
- README.rst: fixed preformatted text (that wasn't)
- Refactored cp tests to work from a common base which more closely resembles the gcp test. All-CVID test input file is in 4-column format (lots of missing data, though)
- Revamp of c to p based on tests results; checkpoint. Sanity & EH tests all run.
- Update makefile to include a mechanism for generating code coverage during tests
- Updated Makefile test task to skip tests prefixed with test_nightly; added task to run all; enabled all cvid test to check this
- add missing files to package_data
- added Apache license and code boilerplate to all source files and scripts
- added architecture & dependency info to README.rst
- added comments to failed and broken tests
- added examples directory
- added sbin/test-runner (see script header for example)
- added setuptools>2.0 to setup.py (testing); updated README.rst
- close branch
- corrected minor README typo
- fix test
- fixed bug in reported AA edit for extensions
- fixed bug introduced in 63e0baf7c986; removed unnecessary and obsolete edti.interface import in tests/framework/mock_input_source.py
- fixed bug that caused protein accession to be not looked up when not specified

- fixed bug with unqualified class names in hgvs.pymeta
- hgvsc to hgvsp bug fixes/updates: changed del/dups to represent the c-terminal end; variants in utr, intron & 1st AA are treated as p.? (subject to review). Cleaned up test data. Tweaked seguid data so the tests pick up the correct NP in a case where there's more than one match - mainly just to get the tests to pass.
- hgvsc to p takes an accession
- make the nightly start from make cleanest (tougher)
- merge into default
- more README and setup.py updates
- move edti bits to bdi
- moved misc/hgvs-shell to sbin
- setup.py: testing yet another dependency_links format
- updated README.rst
- updated bdi and tests to use external UTA instance
- updated examples dir
- updated logo and README

1.11.8 0.0 Series

0.0.9 (2013-12-16)

- added comments to failed and broken tests
- renamed grammars to .pymeta
- consolidated g-c-p testing into a single test file; commented out putatively broken tests; DNAH11 works!
- add forgotten sbin/fasta-seguid for commit -2 (0d29d0ea2d42)
- fixed minor grammar bugs re: AA term and frameshift
- added accession lookup for all of RefSeq protein
- got 'make jenkins' target working
- harmonized with UTA Makefile and setup.py to try to get tests working
- added biopython to setup.py
- fixed pro_eq grammar bug mentioned in #42
- Updated DNAH11 and NEFL tests. They run, so I'll mark as complete, but there are errors associated with the proteins
- hgvsc_to_hgvsp: Fixed a delins bug
- hgvsc_to_hgvsp: Fixed bug in insertion indexing; improved exception handing
- added misc/hgvs-shell to simplify manual testing
- hgvs tests for DNAH11 and NEFL -> note protein not currently working just change if statement
- initial checkin for jenkins branch; want to test this in the build context
- Close branch c_to_p

- Merged in c_to_p (pull request #3)
- Incorporate AASpecial; tests pass.
- merge from default
- merged default into c_to_p
- added AASpecial to handle p.=, p.?, p.0 (and parenthesized versions)
- fixed setup.py issue that caused omission of hgvs.utils on install
- Forgot to add a test file to mercurial
- Merged from default; fixed a test.
- Make test file name more consistent
- SImplified comparison in the event of a simple substitution; updated tests so the failed tests are commented out.
- Reformatted Emily's test data to make it more consumer-friendly; continuous test tweaking - latest checkpoint.
- Another couple of fixes based on EH tests; checking in working version of the tests.
- updated hgvsmapper with all g<->r<->c transformations
- remove explicit class references from makeGrammar invocation, require fully-qualified class name in hgvs.ometa
- close uncertainty branch
- added chr_to_NC in utils, added c_to_g in hgvsmapper
- Name cleanup for tests
- Tests now play nicely with both real data and the mock data.
- Add call to get_tx_seq()
- Missed a rename in the tests.
- Rename test classes to be a bit more consistent with their use.
- Inserted hgvsc_to_hgvsp into hgvsmapper.
- merge from default
- align with developer.rst conventions on naming hgvs variants vs. strings
- Fix tests to run in makefile context; some more documentation
- revamped hgvs_c_to_p so its interface matches hgvsmapper; should make incorporation a simple matter of copying the hgvsc_to_hgvsp method in. Updated tests accordingly. Moved tests to top-level.
- Merge from default
- Re-arranging code for utils/staging for hgvs mapper.
- Purged debug code
- Ack - last checkin broke the tests; fixed accession setup
- format cleanup
- Incorporate stopgap for protein accession; refactor so interface consumes data in the current UTA format; refactor tests to mimic UTA input; getting actual seq is still a placeholder.
- merging default into c_to_p
- added location uncertainty (parsing, representation, formatting, testing)

- added multifastadb code and tests
- [mq]: hgvsmapper-work
- imported patch hgvs-utils-dir
- added multifastadb tool and tests
- added Rudy's AA p.= rule
- [mq]: grammar-relo
- added hgvs.stopgap
- Close branch transcriptmapper
- Merged in transcriptmapper (pull request #2)
- added TODO for tracking, prior to merging pull request
- Basic handling of variants in non-coding regions; will return p.= in all cases; this does not handle the case where a 5'utr variant results in the creation of an upstream Met.
- merged with default, TM bug fixes and more tests
- cleanup names (or at least make them a little more descriptive)
- added tm.cds_start_i in place of hard coding cds
- refactoring
- Roll back exon-specific changes and assume input is entire transcript concatenated together; retain the transcript data as recordtype
- fix test for AA in 2nd exon
- Convert transcript data object to recordtype; add tests for multi-exon (in progress)
- more tests
- additional TM fixes and more tests with multiple exons and strands
- Account for transcripts w/ more than 1 exon (test input assumed one)
- added some 1-exon tests
- Incorporate aa util and extend interval class (for test data); convert code to produce SequenceVariant objects for hgvs c to p. Also hacked in a way to handle p.= into the grammar (should be reviewed before merge).
- bug fixes
- Merged default into c_to_p
- added enum to transcriptmapper tests
- Last cleanup before merging default into here
- all input/output is hgvs-based. updated tests accordingly
- Close branch protein-variants
- Merged in protein-variants (pull request #1)
- hgvs.edit: fixed and improved fs handling, and added mediocre tests
- hgvs.utils: added Xaa=X, Ter=*, Sec=U for aa1-to-aa3 & aa3-to-aa1 translation
- code cleaning
- finished tests for transcriptmapper

- finished all the g,r,c conversions adding more tests
- More cleanup; simplify variant inserter code
- updated transcriptmapper to support g->r, r->g, r->c and appropriate tests
- minor cleanup
- variant insert tests
- merged edti-uta0 branch
- closing branch prior to merge
- edti: added __metaclass__ to edti.interface; added fetch_gene_info to uta0
- hgvs.edti: EDTI base interface and UTA0 implementation milestone
- hgvs.parser: add function attributes for every rule to enable, e.g., Parser.parse_c_interval(...)
- implemented p. parsing and formatting, with tests
- hgvs.utils: handle case when aa string is None
- hgvs.utils: added aa_to_aa{1,3} functions to coerce to 1- or 3-letter amino acids
- hgvs.utils: added protein 1-letter and 3-letter conversion
- Checkpoint for new branch (hgvs c to p)
- branched transcriptmapper
- improved parsing of hgvs_position rules (i.e., without edits) to handle g,m,n,r,c,p types distinctly
- added {gmn,c,r,p}_edit rule to parse variants without accesssions (e.g., c.76A>T)
- renamed DelIns class to RefAlt
- renamed Variant to SequenceVariant, and instance variant seqref to ac
- closed abandoned protein-support branch
- updated parser tests to include aspirational and "reject" tests
- [mq]: import-location-changes
- [mq]: import
- hgvs.location: renamed location classes; added BaseOffset position for r. and c.; removed predicate methods (is_exonic, etc);
- incomplete, buggy milestone
- setup.py: use full path for doc/description.rst
- updated CDSPosition to include datum and added tests
- use get_distribution() rather than require() to fetch version
- Fix for pathing to grammar.txt from within hgvs.parser.Parser
- modified setup.py to zipsafe false
- TODO edited online with Bitbucket
- Making setup.py file pathing absolute
- Fix for setup.py
- updated Makefile and setup.py

- revert directory to current after upload
- fixed bug in HGVSPosition.__str__ and added HGVSPosition test

0.0.7 (2013-10-11)

- fixed bug in HGVSPosition.__str__ and added HGVSPosition test
- collapsed grammar cases for c_pos; fixed variant test case typo

0.0.6 (2013-10-11)

- collapsed grammar cases for c_pos; fixed variant test case typo
- updated docs; fixed typo in variant

0.0.5 (2013-10-11)

- updated docs; fixed typo in variant
- added HGVSPosition (aka HGVS Lite)

0.0.4 (2013-10-11)

- added HGVSPosition (aka HGVS Lite)
- “simple” (single site) variants now pass tests
- update hgvs.__init__ and sphinx to use version from hgtools

0.0.3 (2013-10-10)

- update hgvs.__init__ and sphinx to use version from hgtools
- removed home-grown hg versioning in favor of hgtools
- removed virtualenv support and cleaned up Makefile
- milestone sync; c, gmn, and r types mostly work; some tests broken
- updated variant and added test
- updated grammar (more to do) and tests
- added hgvs.posedit and tests
- updated hgvs.edit
- removed CDSInterval (will use Interval for all intervals)
- fixed typo
- update hgvs.location and tests
- minor setup.py changes

0.0.2 (2013-09-20)

- minor setup.py changes
- grammar simplification; added Laros grammar, examples, comments
- Reverted Lawrence's changes to edit.py (after discussing with him).
- Adding some convenience properties to be used in Geneticus.
- updated grammar; added README.rst
- added missing deps to setup.py; switched to plain ole distutils
- added developer notes, logo, sphinx config

0.0.1 (2014-08-01)

- initial commit

1.12 License

The hgvs package is released under the Apache License 2.0, the text of which appears below:

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.
```

(continues on next page)

(continued from previous page)

```
   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
```

(continues on next page)

(continued from previous page)

```
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
```

(continues on next page)

(continued from previous page)

```
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

F

G

H

I

L

M

N

O

P

R

S

T

U

V