Mutalyzer Documentation

Release 2.0.35

Leiden University Medical Center

Nov 08, 2021

Contents

Mutalyzer is an HGVS variant nomenclature checker. The canonical Mutalyzer installation can be found at mutalyzer.nl.

Note: If you're looking into running your own Mutalyzer installation, you might be interested in *Automated deployment with Ansible*.

This is the developer documentation for Mutalyzer. User documentation can be found on the wiki.

CHAPTER 1

Managing Mutalyzer

Information for getting Mutalyzer running on a system.

1.1 Downloading Mutalyzer

The Mutalyzer2 source code is hosted on GitHub. The recommended way to get the Mutalyzer source code is by cloning the Git repository:

```
git clone https://github.com/mutalyzer/mutalyzer2.git
```

This will give you the current development version. See below for working with other versions.

1.1.1 Release versions

All Mutalyzer releases are tagged. You can run `git tag` to list the available tags. Use `git checkout <tag>` to switch to a certain release. For example, to swith to the *2.0.0* release:

```
git checkout v2.0.0
```

Alternatively, you can checkout the *release* branch. This branch always points to the latest Mutalyzer release.

1.1.2 Archive downloads

If for whatever reason you don't want to use Git, you can download the source code directly as a zip archive or tarball. The current development version can be found from the project homepage. Archive downloads for release versions can be found on the releases page.

1.2 Installation

Note: If you're looking into deploying Mutayzer in a production environment (as opposed to a local development installation), please see *Deploying Mutalyzer in production*.

Mutalyzer depends on a database server, Python 2.7, and several Python packages. Redis is a soft dependency. This section walks you through installing Mutalyzer with Redis and using PostgreSQL as database server, which is the recommended setup.

Note: All operating system specific instructions assume installation on a Debian 8 *jessie* system. You'll have to figure out the necessary adjustments yourself if you're on another system.

The following steps will get Mutalyzer running on your system with the recommended setup:

- *Database server: PostgreSQL*
- *Redis*
- *Python virtual environment*
- *Mutalyzer setup*

At the bottom of this page some *alternative setups* are documented.

1.2.1 If you're in a hurry

The impatient can run Mutalyzer without a database server and more such nonsense with the following steps:

```
$ pip install -r requirements.txt
$ python -m mutalyzer.entrypoints.website
```

This starts the website frontend on the reported port using an in-memory SQLite database.

1.2.2 Database server: PostgreSQL

Install PostgreSQL and add a user for Mutalyzer. Create a database (e.g. `mutalyzer`) owned by the new user. For example:

```
$ sudo apt-get install postgresql
$ sudo -u postgres createuser --superuser $USER
$ createuser --pwprompt --encrypted --no-adduser --no-createdb --no-createrole␣
↪mutalyzer
$ createdb --encoding=UNICODE --owner=mutalyzer mutalyzer
```

Also install some development libraries needed for building the `psycopg2` Python package later and add the package to the list of requirements:

```
$ sudo apt-get install libpq-dev
$ echo psycopg2 >> requirements.txt
```

This will make sure the Python PostgreSQL database adapter gets installed in the *Python virtual environment* section.

See also:

Database server: MySQL Alternatively, MySQL can be used as database server.

Database server: SQLite Alternatively, SQLite can be used as database server.

Dialects – SQLAlchemy documentation In theory, any database supported by SQLAlchemy could work.

1.2.3 Redis

Mutalyzer uses Redis for non-critical fast storage such as statistics:

```
$ sudo apt-get install redis-server
```

Note: Redis is a soft dependency, meaning that Mutalyzer will run without it (but may lack some non-essential features such as caching of external resources).

1.2.4 Python virtual environment

It is recommended to run Mutalyzer from a Python virtual environment, using virtualenv. Installing virtualenv and creating virtual environments is not covered here.

Assuming you created and activated a virtual environment for Mutalyzer, install all required Python packages:

```
$ sudo apt-get install python-dev libmysqlclient-dev libxml2-dev libxslt-dev swig
$ pip install -r requirements.txt
```

Install Mutalyzer:

```
$ python setup.py install
```

Note: If you're planning on modifying the Mutalyzer source code, it might be convenient to install Mutalyzer in *development mode*:

```
$ python setup.py develop
```

Instead of copying the source code to the installation directory, this only links from the installation directory to the source code such that any changes you make to it are directly available in the environment.

Now might be a good time to run the unit tests:

```
$ py.test
```

See also:

virtualenv `virtualenv` is a tool to create isolated Python environments.

virtualenvwrapper `virtualenvwrapper` is a set of extensions to the `virtualenv` tool. The extensions include wrappers for creating and deleting virtual environments and otherwise managing your development workflow.

1.2.5 Mutalyzer setup

Mutalyzer looks for its configuration in the file specified by the `MUTALYZER_SETTINGS` environment variable. First create the file with your configuration settings, for example:

```
$ export MUTALYZER_SETTINGS=~/mutalyzer/settings.py
$ cat > $MUTALYZER_SETTINGS
REDIS_URI = 'redis://localhost'
DATABASE_URI = 'postgresql://mutalyzer:*****@localhost/mutalyzer'
```

A script is included to setup the database:

```
$ mutalyzer-admin setup-database --alembic-config migrations/alembic.ini
```

You can now proceed to *Running Mutalyzer*.

See also:

Configuration For more information on the available configuration settings.

1.2.6 Alternative setups

The remainder of this page documents some alternatives to the recommended setup documented above.

Database server: MySQL

Install MySQL and create a database (e.g. `mutalyzer`) with all privileges for the Mutalyzer user. For example:

```
$ sudo apt-get install mysql-server
$ mysql -h localhost -u root -p
> create database mutalyzer;
> grant all privileges on mutalyzer.* to mutalyzer@localhost identified by '*****';
```

The Python MySQL database adapter is a hard dependency regardless of your choice of database server, so it'll get installed in the *Python virtual environment* section.

In the *Mutalyzer setup* section, make sure to use a MySQL database URI in the Mutalyzer settings file, e.g.:

```
DATABASE_URI = 'mysql://mutalyzer:*****@localhost/mutalyzer?charset=utf8'
```

See also:

Database server: PostgreSQL The recommended setup uses PostgreSQL as database server.

Database server: SQLite

You probably already have all you need for using SQLite, so this section consists of zero steps.

Just note that in the *Mutalyzer setup* section, you should use an SQLite database URI in the Mutalyzer settings file, e.g.:

```
DATABASE_URI = 'sqlite:////tmp/mutalyzer.db'
```

See also:

Database server: PostgreSQL The recommended setup uses PostgreSQL as database server.

1.3 Configuration

This section describes how to configure Mutalyzer and includes a list of all available configuration settings.

Mutalyzer looks for its configuration in the file specified by the `MUTALYZER_SETTINGS` environment variable. Make sure to always have this environment variable set when invoking any component of Mutalyzer. One way of doing this is by exporting it:

```
$ export MUTALYZER_SETTINGS=~/mutalyzer/settings.py
```

If you like, you can add this command to your `~/.bashrc` to have it executed every time you open a shell.

Another way is by prefixing your invocations with `MUTALYZER_SETTINGS=...`. For example:

```
$ MUTALYZER_SETTINGS=~/mutalyzer/settings.py mutalyzer-website
```

1.3.1 Example configuration

If you followed the steps in *Installation*, this is a standard configuration file that will work for you:

```
REDIS_URI = 'redis://localhost'
DATABASE_URI = 'postgresql://mutalyzer:*****@localhost/mutalyzer'
```

This is not yet a minimal configuration. In fact, you can run Mutalyzer without a configuration file since the default configuration works out of the box. The default configuration uses a mock Redis instance, an in-memory database backend and a temporary directory for cache and log files, so it is not recommended for anything more than playing around.

The next section describes all available configuration settings.

1.3.2 Configuration settings

Note that the configuration file is interpreted as a Python module, so you can use arbitrary Python expressions as configuration values, or even import other modules in it.

Unsetting a configuration setting is done by using the value *None*. If no default value is mentioned for any configuration setting below it means it is not set by default.

EMAIL The email address used in contact information on the website and sent with NCBI Entrez calls.

Default value: `info@mutalyzer.nl`

BATCH_NOTIFICATION_EMAIL The email address used as sender in batch job notifications. If set to *None*, the value of *EMAIL* will be used.

Default value: None

DEBUG If set to *True*, Mutalyzer runs in debug mode and will show more information with errors.

Default value: False

CACHE_DIR The cache directory which is used to store uploaded and downloaded files such as reference files from the NCBI and batch job results.

Default value: `/tmp`

User input settings

MAX_FILE_SIZE Maximum size for uploaded and downloaded files (in bytes).

*Default value: 10 * 1048576* (10 MB)

EXTRACTOR_MAX_INPUT_LENGTH Maximum sequence length for description extractor (in bases).

*Default value: 50 * 1000* (50 Kbp)

BATCH_JOBS_ERROR_THRESHOLD Allow for this fraction of errors in batch jobs.

Default value: 0.05

Database settings

DATABASE_URI SQLAlchemy database connection URI specifying the database used to store users, samples, variants, etcetera.

Database system	Example URI
PostgreSQL	`postgresql://mutalyzer:*****@localhost/mutalyzer`
MySQL	`mysql://mutalyzer:*****@localhost/mutalyzer?charset=utf8`
SQLite	`sqlite:////tmp/mutalyzer.db`

See the SQLAlchemy documentation on Engine Configuration for more information.

Default value: `sqlite://` (in-memory SQLite database)

REDIS_URI Redis connection URI (can be any redis-py connection URI). Set to *None* to silently use a mock Redis. Redis is only used for non-essential features such as caching of external resources.

Default value: None

Settings for output and logging

All Mutalyzer messages come with a level which can be one of:

Level	Alias	Meaning
-1	Log	Specifically log a message.
0	Debug	Debug information.
1	Info	Info.
2	Warning	Regular warnings.
3	Error	Serious errors that can be compensated for.
4	Fatal	Errors that are not recoverable.
5	Off	Can be used as a log/output level to turn off output.

LOG_FILE Name and location of the log file.

Default value: `/tmp/mutalyzer.log`

LOG_LEVEL Level of logged messages.

Default value: 3

OUTPUT_LEVEL Level of output messages.

Default value: 1

LOG_TIME_FORMAT Format of time prefix for log messages. Can be anything that is accepted as the format argument of time.strftime.

Default value: `%Y-%m-%d %H:%M:%S`

Website settings

REVERSE_PROXIED If set to *True*, the WSGI application runs behind a reverse proxy (e.g., nginx using `proxy_pass`). This needs to be set if the application is mapped to a URL other than / or a different HTTP scheme is used by the reverse proxy.

Default value: False

WEBSITE_ROOT_URL URL to the website root (without trailing slash). Used for generating download links in the batch scheduler.

Default value: None

SOAP_WSDL_URL URL to the SOAP webservice WSDL document. Used to build the WSDL document and for linking to it from the documentation page on the website.

Default value: None

JSON_ROOT_URL URL to the HTTP/RPC+JSON webservice root (without trailing slash). Used for linking to it from the documentation page on the website.

Default value: None

Piwik settings

Piwik is an Open Source analytics platform. Mutalyzer has built-in support for visitor tracking with Piwik.

PIWIK If set to *True*, Piwik is enabled and some Javascript tracking code is included in every Mutalyzer website page.

Default value: False

PIWIK_BASE_URL Base URL for the Piwik server.

Default value: `https://piwik.example.com`

PIWIK_SITE_ID Piwik site ID for Mutalyzer.

Default value: 1

Miscellaneous settings

LRG_PREFIX_URL Prefix URL from where LRG files are fetched.

Default value: `ftp://ftp.ebi.ac.uk/pub/databases/lrgex/SCHEMA_1_7_ARCHIVE/`

DEFAULT_ASSEMBLY Default genome assembly (by name or alias).

Default value: `hg19`

NEGATIVE_LINK_CACHE_EXPIRATION Cache expiration time for negative transcript<->protein links from the NCBI (in seconds).

*Default value: 60 * 60 * 24 * 30* (30 days)

USE_RELOADER Enable the Werkzeug reloader for the website.

This is disabled by default due to a bug with using the reloader in combination with `python -m mutalyzer.entrypoints.website`.

Default value: False

1.4 Running Mutalyzer

Please make sure Mutalyzer can find its configuration file, as detailed in *Configuration*.

Mutalyzer comes with a number of different interfaces, of which the website is perhaps the main one. It can be started using a built-in test server that's useful for development and debugging purposes like this:

```
$ mutalyzer-website
 * Running on http://127.0.0.1:5000/
```

You can now point your webbrowser to the URL that is printed and see the welcoming Mutalyzer homepage.

Likewise, the SOAP and HTTP/RPC+JSON webservices can be started with the `mutalyzer-service-json` and `mutalyzer-service-soap` commands, respectively.

For processing batch jobs, the batch processor must be running. This process can be started from the command line and will keep running until it is stopped by pressing Ctrl+C:

```
$ mutalyzer-batch-processor
^Cmutalyzer-batch-processor: Hitting Ctrl+C again will terminate any running job!
mutalyzer-batch-processor: Graceful shutdown
```

The built-in test servers won't get you far in production, though, and there are many other possibilities for deploying Mutalyzer using WSGI. This topic is discussed in *Deploying Mutalyzer in production*.

1.5 Upgrading

Before upgrading Mutalyzer, stop any currently running instances. Then, update your copy of the source code (using for example `git pull` on an existing git clone).

Make sure to install any new requirements:

```
$ pip install -r requirements.txt
```

Now install the new version:

```
$ python setup.py install
```

Managing database migrations is done using Alembic. This command will move your database to the latest schema:

```
$ alembic -c migrations/alembic.ini upgrade head
```

1.6 Administration

For administrative tasks, Mutalyzer comes with the `mutalyzer-admin` command line utility. Use its `-h` argument in combination with any subcommand for detailed usage information, for example:

```
$ mutalyzer-admin setup-database -h
usage: mutalyzer-admin setup-database [-h] [--destructive] [-c ALEMBIC_CONFIG]

Setup database tables (if they do not yet exist).

optional arguments:
  -h, --help            show this help message and exit
  --destructive         delete any existing tables and data
  -c ALEMBIC_CONFIG, --alembic-config ALEMBIC_CONFIG
                        path to Alembic configuration file

If Alembic config is given (--alembic-config), this also prepares the database
for future migrations with Alembic (recommended).
```

1.6.1 Managing genome assemblies

Mutalyzer can be loaded with any number of genome assemblies. Each assembly includes a list of chromosomes. To list the currently loaded genome assemblies:

```
$ mutalyzer-admin assemblies list
GRCh37 (hg19), Homo sapiens (9606)
GRCm38 (mm10), Mus musculus (10090)
```

Loading a new genome assembly is done with information in a JSON file, of which there are some examples in the Mutalyzer source tree under the `extras/assemblies` directory, for example:

```
$ mutalyzer-admin assemblies add extras/assemblies/GRCh37.json
```

For any genome assembly, transcript mappings can be imported. These include genomic coordinate mappings of their CDS and exons. Currently, three sources of transcript mappings are supported.

Note: The following `mutalyzer-admin assemblies` subcommands all accept an optional `--assembly` argument to specify the genome assembly to import to.

Import mappings from an NCBI mapview file

The NCBI provides FTP downloads of transcript mappings for a large number of genome assemblies as used by their Map Viewer service. These can be imported with `mutalyzer-admin`, but only after sorting by the *feature_id* and *chromosome* columns.

For example, to import transcript mappings for the GRCh37 assembly, run the following:

```
$ wget ftp://ftp.ncbi.nlm.nih.gov/genomes/MapView/Homo_sapiens/sequence/ANNOTATION_
↪RELEASE.105/initial_release/seq_gene.md.gz
$ zcat seq_gene.md.gz | sort -t $'\t' -k 11,11 -k 2,2 > seq_gene.sorted.md
$ mutalyzer-admin assemblies import-mapview seq_gene.sorted.md 'GRCh37.p13-Primary␣
↪Assembly'
```

Note: The last argument, `GRCh37.p13-Primary Assembly`, defines the group label to filter the file on. You would usually want to include it.

Examples for other assemblies can be found in this Gist.

Import mappings from an EBI LRG transcripts map file

The EBI provides FTP downloads of transcript mappings for all of the LRG sequences on the latest human genome assembly. These can be imported with `mutalyzer-admin`.

For example, to import LRG transcript mappings for the GRCh37 assembly, run the following:

```
$ wget ftp://ftp.ebi.ac.uk/pub/databases/lrgex/list_LRGs_transcripts_GRCh37.txt -O /
↪tmp/hg19.lrgmap.txt
$ mutalyzer-admin assemblies import-lrgmap -a hg19 /tmp/hg19.lrgmap.txt
```

Import mappings from the UCSC Genome Browser MySQL database

Transcript mappings from the UCSC Genome Browser MySQL database can be imported on a per-gene basis. This is useful when the NCBI mappings do not (yet) include a certain gene or transcript.

For example, to import all TTN transcript mappings:

```
$ mutalyzer-admin assemblies import-gene TTN
```

Note: This subcommand chooses the UCSC genome assembly by using the alias of the specified Mutalyzer genome assembly (*hg19* by default).

Import mappings from a reference file

For transcript mappings that are not available from our usual sources, importing from a genomic reference is supported:

```
$ mutalyzer-admin assemblies import-reference NC_012920.1
```

Note: Currently this subcommand is restricted to importing mtDNA transcripts, since it has the chromosome hard coded and only supports one exon per transcript.

1.6.2 Showing announcements to users

It is possible to define an announcement to be shown on the website interface. For example, to display *Hello World!* with a link to the GNU Hello World! page:

```
$ mutalyzer-admin announcement set 'Hello World!' \
    --url http://www.gnu.org/fun/jokes/helloworld.html
```

To remove the announcement, use `unset`:

```
$ mutalyzer-admin announcement unset
```

1.6.3 Synchronizing the cache with other installations

Using the `sync-cache` subcommand, the reference file cache of a remote Mutalyzer installation can be queried for new entries which are then retrieved and added to the local cache.

The primary purpose for this is synchronizing reference files loaded by users with the reference file loader between different servers. These reference files are assigned a unique accession number (starting with `UD_`) upon creation, which is at that point unknown to any other Mutalyzer server.

For example, to synchronize the local reference file cache with the primary Mutalyzer server:

```
$ mutalyzer-admin sync-cache 'https://mutalyzer.nl/services/?wsdl' \
    'https://mutalyzer.nl/Reference/{file}'
```

1.6.4 Mutalyzer database setup

After installation, a database needs to be setup for Mutalyzer to run (see *Mutalyzer setup*):

```
$ mutalyzer-admin setup-database --alembic-config migrations/alembic.ini
```

The `--alembic-config` argument points to the `alembic.ini` file in the Mutalyzer source tree and it enables initialization of database migration management. It is recommended to include it, but you don't need it if you don't plan to ever upgrade your Mutalyzer installation.

This subcommand also takes an optional `--destructive` argument, which can be used to remove any existing database content.

1.7 Deploying Mutalyzer in production

The previous sections discussed managing a Mutalyzer installation with a focus on a development environment. There are a number of additional things you will want to consider when deploying Mutalyzer in a production environment, mainly concerning security and performance.

Usually you'll at least want to use a well-performing WSGI application server for the website and SOAP and HTTP/RPC+JSON webservices. There are many options here, ranging from Apache's mod_wsgi to uWSGI to standalone WSGI containers such as Gunicorn.

Below we briefly describe our recommended setup for a production environment using Gunicorn, nginx and Supervisor.

See *Automated deployment with Ansible* for a solution to completely automate this.

1.7.1 Configuration settings

At a minimum, it is recommended to have a look at the following configuration settings:

- *EMAIL*
- *DEBUG*
- *CACHE_DIR*
- *WEBSITE_ROOT_URL*
- *SOAP_WSDL_URL*
- *JSON_ROOT_URL*
- *REVERSE_PROXIED*

1.7.2 WSGI application server: Gunicorn

Gunicorn is a well-perfoming Python WSGI HTTP Server. Being a Python application, it can be installed in the Mutalyzer virtual environment with `pip install gunicorn`.

Many configuration settings are available for Gunicorn and we recommend to use a configuration file per WSGI application. For example, the following configuration can be stored in `website.conf`:

```
workers = 4
max_requests = 1000
timeout = 600
bind = 'unix:/opt/mutalyzer/run/website.sock'
```

This will bind the Gunicorn server to a unix socket (which we can later use from nginx) and run with 4 worker processes. To serve the Mutalyzer website with this configuration, run the following:

```
$ gunicorn -c website.conf mutalyzer.entrypoints.website
```

This uses the WSGI application object exported by the *mutalyzer.entrypoints.website* module. Likewise, the SOAP and HTTP/RPC+JSON webservices have WSGI application objects exported by the *mutalyzer.entrypoints.service_soap* and *mutalyzer.entrypoints.service_json* modules.

1.7.3 Web server: nginx

It is usually a good idea to use a separate webserver in front of the WSGI application servers. We use nginx for this purpose and configure it to server static files directly and act as a reverse proxy for the WSGI applications.

For example, to serve the website from the root path and the HTTP/RPC+JSON webservice from the `/json` path, an nginx configuration similar to the following can be used:

```
server {
    listen                      80;
    server_name                 _;

    client_max_body_size        2G;
    keepalive_timeout           5;

    location /static/ {
        alias /opt/mutalyzer/static/;
        expires 30d;
        add_header Pragma public;
        add_header Cache-Control "public";
    }

    location / {
        root                    /usr/share/nginx/html;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Real-IP $remote_addr;
        proxy_set_header        X-Scheme $scheme;
        proxy_set_header        Host $http_host;
        proxy_redirect          off;
        proxy_read_timeout      600;
        proxy_pass              http://website;
    }

    location /json {
```

(continues on next page)

(continued from previous page)

```
        root                    /usr/share/nginx/html;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Real-IP $remote_addr;
        proxy_set_header        X-Scheme $scheme;
        proxy_set_header        Host $http_host;
        proxy_redirect          off;
        proxy_read_timeout      600;
        proxy_pass              http://service-json;
    }
}

upstream website {
    server                      unix:/opt/mutalyzer/run/website.sock fail_timeout=0;
}

upstream service-json {
    server                      unix:/opt/mutalyzer/run/service-json.sock fail_
↪timeout=0;
}
```

1.7.4 Process control: Supervisor

For managing the different WSGI application servers and Mutalyzer batch processor, Supervisor can be used. Supervisor is usually started from the init system and controls programs and program groups. For example, it can automatically restart a program if it crashed for some reason.

The following is an example Supervisor configuration defining a Mutalyzer group consisting of the batch processor and a Gunicorn process for the website:

```
[group:mutalyzer]
programs=batch-processor,website

[program:batch-processor]
command=mutalyzer-batch-processor
autorestart=true
environment=MUTALYZER_SETTINGS="/opt/mutalyzer/conf/settings.py"

[program:website]
command=gunicorn -c /opt/mutalyzer/conf/website.conf mutalyzer.entrypoints.website
autorestart=true
environment=MUTALYZER_SETTINGS="/opt/mutalyzer/conf/settings.py"
```

1.7.5 Automated deployment with Ansible

Deployments of complete production environments are often complex and repetitive. Therefore, manual deployments are inefficient and error-prone. Several systems exist to automate this, such as Puppet, Chef, and Ansible.

An automated deployment of Mutalyzer with Ansible is available on GitHub. This includes installation of the website, SOAP and HTTP/RPC+JSON webservices, and the batch processor, similar to the setup described above.

CHAPTER 2

Development

General information for Mutalyzer developers.

2.1 Contributing

Contributions to Mutalyzer are very welcome! They can be feature requests, bug reports, bug fixes, unit tests, documentation updates, or anything els you may come up with.

Development of Mutalyzer happens on GitHub: https://github.com/mutalyzer/mutalyzer2

2.1.1 Coding style

In general, try to follow the PEP 8 guidelines for Python code and PEP 257 for docstrings.

2.1.2 Installation

As a developer, you probably want to install the Mutalyzer package in development mode. This will allow you to edit files directly in the source directory without having to reinstall.

Please refer to *Installation* for general installation instructions. For development mode installation, instead of using `python setup.py install`, use:

```
python setup.py develop
```

2.1.3 Creating a pull request

Contributions are most welcome as GitHub pull requests. If you're familiar with the typical GitHub pull request workflow, you can skip this section.

New features are best implemented in their own branches, isolating the work from unrelated developments. In fact, it's good practice to *never work directly on the master branch* but always in a separate branch. When your work is ready, a feature branch can be merged back into master via a *pull request* in GitHub.

Before starting your work, fork the Mutalyzer repository to your own namespace in GitHub and work from this fork. Before starting work on your feature, create a branch for it:

```
git clone https://github.com/<you>/mutalyzer.git
cd mutalyzer
git checkout -b your-feature
```

Commit changes on this branch. If you're happy with it, push to GitHub:

```
git push origin your-feature -u
```

Now create a pull request to discuss the implementation with the Mutalyzer maintainers. This might involve adding additional commits which are included in the pull request by pushing your branch again:

```
git commit
git push
```

If the work is done, a maintainer can merge your branch and close the pull request. After the branch was merged you can safely delete it:

```
git branch -d your-feature
```

2.2 Testing

2.2.1 Unit tests

We use pytest for the unit tests. To run them, just type `py.test` from the Mutalyzer source directory.

Note: The Mutalyzer package must be installed before running the unit tests.

By default, the tests use an in-memory SQLite database. This can be customized with the `--database-uri` command line argument and a valid SQLAlchemy connection URI (obviously, the contents of this database will be lost). For example, to use an SQLite database on the filesystem:

```
$ py.test --database-uri sqlite:////tmp/mutalyzer.sql
```

Or, using pg_virtualenv (included with the Debian PostgreSQL packages), to run the tests with PostgreSQL:

```
$ pg_virtualenv bash -c 'py.test --database-uri postgres://${PGUSER}:${PGPASSWORD}@$
↪{PGHOST}:${PGPORT}/${PGDATABASE}'
```

Multiple `--database-uri` arguments are allowed. Tests using the database will be run once for every database specified.

Similarly, `--redis-uri` (only one allowed) specifies a Redis server to use for testing. If unspecified, a mock Redis server is used.

Tests are run automatically on Travis CI with SQLite, PostgreSQL, and MySQL, for each pull request and push on GitHub.

2.2.2 Testing the web services

To ease testing the web services during development, some simple web service client scripts are included in the Mutalyzer source tree:

```
$ cd extras/soap-tools
$ ./info.py
Version: 2.0.31
Version parts: 2, 0, 31
Release date: 21 August 2019
Nomenclature version: 2.0
Nomenclature version parts: 2, 0
Server name: res-muta-app01
Contact e-mail: info@mutalyzer.nl
$
```

They simply call one of the web service functions and print the result. You may have to change the server location defined at the top of these scripts.

Note: One of the scripts, `run_batch_job.py`, provides an easy way to run a batch job from the command line. Some additional notes are available for running this on a Windows machine.

2.3 Releases

2.3.1 Versioning

A normal version number takes the form X.Y.Z where X is the major version, Y is the minor version, and Z is the patch version. Development versions take the form X.Y.Z.dev where X.Y.Z is the closest future release version.

Note that this scheme is not 100% compatible with SemVer which would require X.Y.Z-dev instead of X.Y.Z.dev but compatibility with setuptools is more important for us. Other than that, version semantics are as described by SemVer.

Releases are available from the GitHub git repository as tags. Additionally, the latest release is available from the *release* branch.

Note: Older Mutalyzer version numbers took the form 2.0.beta-X where X was incremented on release.

2.3.2 Release procedure

Releasing a new version is done as follows. This assumes remote *github* is the upstream Mutalyzer repository and you have push rights there.

1. Start a release branch and make sure the section in the `CHANGES.rst` file for this release is complete:

   ```
   git checkout -b release-X.Y.Z
   git add CHANGES.rst
   git commit -m 'Update changelog'
   ```

   Note: Commits since release X.Y.Z can be listed with `git log vX.Y.Z..` for quick inspection.

   ---

2. Update the `CHANGES.rst` file to state the current date for this release and edit `mutalyzer/__init__.py` by updating *__date__* and removing the `dev` value from *__version_info__*.

   Commit and tag the version update:

   ```
   git commit -am 'Bump version to X.Y.Z'
   git tag -a 'vX.Y.Z'
   ```

3. Add a new entry at the top of the `CHANGES.rst` file like this:

   ```
   Version X.Y.Z+1
   ---------------

   Release date to be decided.
   ```

   Increment the patch version and add a `dev` value to *__version_info__* in `mutalyzer/__init__.py` and commit these changes:

   ```
   git commit -am 'Open development for X.Y.Z+1'
   ```

4. Push these commits to GitHub:

   ```
   git push github release-X.Y.Z -u
   ```

   And submit a pull request for this branch.

5. If everything looks ok and the pull request has been accepted you can push the tag and update the *release* branch. Make sure to re-tag if the pull request was updated meanwhile. The release branch can be deleted.

   ```
   git push github +vX.Y.Z~0:refs/heads/release --tags
   git branch -d release-X.Y.Z
   ```

   That last push command might seem a bit cryptic (it is explained here). It sets the remote branch *release* to whatever the tag *vX.Y.Z* points to and also pushes all tags.

   If the *release* branch is protected, updating it will be rejected until the required status checks pass (e.g., Travis CI).

2.4 Database migrations

Managing database migrations is done using Alembic, a database migration tool for usage with SQLAlchemy.

2.4.1 Supported database systems

We try to take care to fully support migrations on PostgreSQL, MySQL, and SQLite. Since our production deployments run PostgreSQL, migrations may be optimized only for that system.

2.4.2 Zero-downtime migrations

To support zero-downtime database migrations, all migrations must keep compatibility with the existing codebase. The assumption is that deployments always run database migrations first and update the code second.

This means some database changes may have to be broken down into several steps and completed over several deployments. For example, deleting a column from a table can be done as follows:

1. Remove the code that uses the column.
2. Deploy:
    1. Run migrations (nothing to be done).
    2. Update the code.
3. Create a migration that removes the column.
4. Deploy:
    1. Run migrations (column is removed).
    2. Update the code (nothing to be done).

On the other hand, adding a column is easier. Creating a migration that adds the column and adding code to use it can safely be done at the same time if we assume migrations are always run before updating the code.

To support smooth deployments, ideally a release is made between migrations and code updates that should be deployed separately. This way also external users can always safely upgrade one release at a time.

Note: Our automated deployment of Mutalyzer with Ansible runs database migrations before updating the code.

2.4.3 Setting up Alembic

Before you can use Alembic, the database needs to be stamped with the current revision. This is done automatically when using the `-c` argument with the `mutalyzer-admin setup-database` command (as is recommended in *Mutalyzer setup*):

```
$ mutalyzer-admin setup-database --alembic-config migrations/alembic.ini
```

An existing database can also be stamped manually using Alembic:

```
$ alembic -c migrations/alembic.ini stamp head
```

2.4.4 Running migrations

Upgrading an existing database to the latest revision is done as follows:

```
$ alembic -c migrations/alembic.ini upgrade head
```

Downgrades are explicitely unsupported and some migrations may not have downgrades implemented at all.

2.4.5 Creating migrations

To create a new migration, first update the SQLAlchemy models in the Mutalyzer source code. Then, generate a migration with Alembic:

```
$ alembic -c migrations/alembic.ini revision --autogenerate -m 'Some descriptive␣
↪comment'
```

Template code for the upgrade and downgrade paths are written to a new file in the `migrations/versions` directory. Alembic is smart enough to generate the complete code for simple migrations, but please always have a look at the generated code.

For more complex changes to the database schema, you'll have to add some code manually. The same goes for any data migrations you might want to include. Consult the Alembic documentation and existing migrations for some common patterns.

2.5 String representations

We live in a global economy with many different languages and alphabets. Using byte strings for text and just assuming everything is ASCII encoded is suboptimal and *will* lead to bugs. These bugs may even be security issues.

That's why Mutalyzer uses unicode strings wherever possible and tries to be aware of encodings when dealing with input and output. Here we describe how we do it.

2.5.1 String representations in Python

Since Mutalyzer only runs on Python 2.7, we can ignore all older Python versions and Python 3. So, the two main string types in Python are:

1. *str*, byte strings
2. *unicode*, unicode strings

Byte strings are the default string type in Python 2.7 and are for example the type you get when writing a string literal:

```
>>> type('mutalyzer')
<type 'str'>
```

Unicode string literals can be written using the `u` prefix:

```
>>> type(u'mutalyzer')
<type 'unicode'>
```

Many modules from the Python standard library and also third party libraries consume and produce byte strings by default and may or may not work correctly with unicode strings.

2.5.2 Unicode strategy

Internally, all strings should be represented by unicode strings as much as possible. The main exceptions are large reference sequence strings. These can often better be BioPython sequence objects, since that is how we usually get them in the first place.

Our main strategy is as follows:

1. We use `from __future__ import unicode_literals` at the top of every file.
2. All incoming strings are decoded to unicode (if necessary) as soon as possible.
3. Outgoing strings are encoded to UTF8 (if necessary) as late as possible.
4. BioPython sequence objects can be based on byte strings as well as unicode strings.
5. In the database, everything is UTF8.
6. We must be aware of the encoding of files supplied by the user or downloaded from external sources.

Point 1 ensures that all string literals in our source code will be unicode strings:

```
>>> from __future__ import unicode_literals
>>> type('mutalyzer')
<type 'unicode'>
```

As for point 4, sometimes this may even change under our eyes (e.g., calling `.reverse_complement()` will change it to a byte string). We don't care as long as they're BioPython objects, only when we get the sequence out we must have it as unicode string. Their contents are always in the ASCII range anyway.

Although *Bio.Seq.reverse_complement* works fine on Python byte strings (and we used to rely on that), it crashes on a Python unicode string. So we take care to only use it on BioPython sequence objects and wrote our own reverse complement function for unicode strings (*mutalyzer.util.reverse_complement*).

2.5.3 Files

The Python builtin open cannot decode file contents and just yields byte strings. Therefore, we typically use io.open instead, which accepts an *encoding* argument.

Downloaded reference files are stored UTF8 encoded (and then bzipped). We can assume UTF8 encoding when reading them back from disk.

We try to detect the encoding of user uploaded text files (batch jobs, GenBank files) and assume UTF8 if detection fails.

2.5.4 Libraries

SQLAlchemy, our database toolkit, transparently sends both byte strings and unicode strings UTF8 encoded to the database and presents all strings as unicode strings to us.

The webframework Mutalyzer uses, Flask, is also fully unicode based.

The Mutalyzer webservices are based on Spyne. The Spyne documentation has the following to say about its *String* and *Unicode* types:

> There are two string types in Spyne: *spyne.model.primitive.Unicode* and *spyne.model.primitive.String* whose native types are *unicode* and *str* respectively.
>
> Unlike the Python *str*, the Spyne *String* is not for arbitrary byte streams. You should not use it unless you are absolutely, positively sure that you need to deal with text data with an unknown encoding. In all other cases, you should just use the *Unicode* type. They actually look the same from outside, this distinction is made just to properly deal with the quirks surrounding Python-2's *unicode* type.
>
> Remember that you have the *ByteArray* and *File* types at your disposal when you need to deal with arbitrary byte streams.
>
> The *String* type will be just an alias for *Unicode* once Spyne gets ported to Python 3. It might even be deprecated and removed in the future, so make sure you are using either *Unicode* or *ByteArray* in your interface definitions.

So let's not ignore that and never use *String* in our webservice interface.

The pyparsing library is used for parsing HGVS variant descriptions. Overall it can deal with unicode input and also yields unicode output in that case. However, there are some exceptions where we explicitely have to decode to a unicode string (for example, omitted optional parts yield the empty byte string).

2.5.5 Python 3

The situation in Python 3 is very different from Python 2.7. The two main string types in Python 3 are:

1. *str*, unicode strings
2. *byte*, byte strings

Unicode strings are the default string type in Python 3 and are for example the type you get when writing a string literal:

```
>>> type('mutalyzer')
<class 'str'>
```

Byte string literals can be written using the `b` prefix:

```
>>> type(b'mutalyzer')
<class 'bytes'>
```

Many modules from the Python standard library and also third party libraries consume and produce unicode strings by default and may or may not work correctly with byte strings.

What does this mean for Mutalyzer? Actually, our current approach takes us quite a bit closer to how things are generally done in Python 3. However, Mutalyzer is very much not Python 3 compatible, even the unicode handling parts are only valid in Python 2.7 on some points.

CHAPTER 3

Additional notes

3.1 Adding a new organism to Mutalyzer

3.1.1 Introduction

In this document, we describe what is needed for Mutalyzer to support new organisms. For each functionality, we give a list of requirements and an estimate of the amount of time needed for implementation.

3.1.2 Position converter

To use the position converter, we need a mapping database for the genomic reference sequence that is used. The database should contain the following information per gene:

- Genomic location (chromosome, transcription start, transcription end).
- Gene model (CDS start, CDS stop, all splice sites).

The transcription start and end may be missing, but this is not recommended.

Implementation time

Depending on the format of the database, making this functionality available is relatively straightforward. If the database is stored in a structured format (CSV or something that can be converted to CSV automatically), importing should take no more than two working days.

3.1.3 Name checker

To use the name checker, a fully annotated GenBank record should be available for every genomic location.

Public genome build

If the genome build of the organism in question is public, as is the case for some model organisms, the GenBank records can be retrieved from the NCBI. In this case, no special effort is required.

If the data is public, but is not yet available as a genome build at the NCBI, the mapping database can be converted to the format required by the NCBI and uploaded. The conversion should take no more than two working days, but the time it will take before the build is available depends on the NCBI.

Non public genome build

If it is not possible to submit the generated GenBank reference files to the NCBI, for commercial reasons for example, there is the option to offer a stand-alone version of Mutalyzer containing the reference files. This version should be hosted at the client side. Preparing such an installation will require no more than three working days.

3.1.4 Additional notes

There is currently no program available for the generation of the GenBank reference files. Although this is a one time effort, we estimate that the development of such a program requires three weeks.

There is also no program available for the conversion of a mapping database to the format the NCBI expects. The specifications are also unknown. Once we have the specifications, we estimate that the development of a conversion tool will take three weeks.

A stand-alone version of Mutalyzer will require some expertise to set up and maintain at the client side. Although we currently provide no support or maintenance, we are thinking about a model for this.

3.2 Changelog

This is a record of changes made between each Mutalyzer release.

3.2.1 Version 2.0.35

Released on November 8th 2021.

- Rename repository links (#526).

3.2.2 Version 2.0.34

Released on March 15th 2021.

- Update article link and point towards the new release website (#518).

3.2.3 Version 2.0.33

Released on November 24th 2020.

- Update links (#514).
- Improve warning messages (#513).
- Add legacy crossmapper module (#508).

3.2.4 Version 2.0.32

Released on December 9th 2019.

- Update email address and wiki link (#498).
- Fix erroneous message for mtDNA m. coordinate system (#497).
- Update requirements (#496).
- Fix invalid LRG record issue (#495).

3.2.5 Version 2.0.31

Released on August 21st 2019.

- Update requirements (#486).
- Add support for NCBI E-utilities API key (#485).
- Update HGVS descriptions retrieval for SNP converter (#482)
- Make file selection mandatory in the batch job web page (#481).
- Update Travis configuration (#483)

3.2.6 Version 2.0.30

Released on May 24th 2019.

- Update requirements (#475).
- Update documentation links (#474).
- Fix the genbank parser to perform loci objects creation in two steps (#468).

3.2.7 Version 2.0.29

Released on December 4th 2018.

- Improve user feedback for incorrect specific locus (#459).
- Improve some position converter warning (#458).
- Update requirements (#457).
- Make API more consistent between UDs and NCs (#456).
- Fix internal server error on IVS variant (#452).

3.2.8 Version 2.0.28

Released on June 11th 2018.

- Support for LRG XML schema version 1.9 (#449).

3.2.9 Version 2.0.27

Released on May 25th 2018.

- Accept variant descriptions with NC files as references (#445).
- Removed support for pending LRG reference files (#444).
- Add link to wiki page with differences between Mutalyzer and HGVS (#443).
- Fix for batch processor crash due to NCBI server timeout (#442).
- Update genbank retriever parameters according to NCBI recommendations (#439).
- Update getGeneLocation‘ to use standard strategy to select mapping (#437).
- Add link to website changelog (#434).

3.2.10 Version 2.0.26

Released on July 19th 2017.

- Description-extractor dependency updated to version 2.35 (#429).
- Fix for negative cCDSStop due to wrong transcript to protein link (#430).

3.2.11 Version 2.0.25

Released on May 17th 2017.

- Fix for batch processor crash when trying to alter an “item” column of the “batch_queue_items” database table (#426).

3.2.12 Version 2.0.24

Released on April 12th 2017.

- Fix for SNP converter crash when called with ‘rs0’ as parameter. SNP converter displays now more warning messages (#419).

3.2.13 Version 2.0.23

Released on November 9th 2016.

- Fix for internal server error in the position converter triggered by non-variant description (#333).

3.2.14 Version 2.0.22

Released on September 20th 2016.

- Biopython dependency updated to version 1.68 (#411).

3.2.15 Version 2.0.21

Released on June 24th 2016.

- Fix typo in batch SNP converter, causing all batch jobs to stall (#409).

3.2.16 Version 2.0.20

Released on June 22th 2016.

- Accept accession number as transcript selector (#405).
- Add legend to the output of runMutalyzer service (#404).
- Remove support for GI numbers (#396).

3.2.17 Version 2.0.19

Released on June 9th 2016.

- Fix reference file loading by URL (#383).
- Consider accession version in *getGeneName* webservice method (#390).

3.2.18 Version 2.0.18

Released on May 23rd 2016.

- Fixed installation on CentOS 6 (#358).
- Auto focus primary form input on page load (#363).
- Check optional argument for `dup` (#369).
- Fixes for chromosome slicing by gene symbol (#371).

3.2.19 Version 2.0.17

Released on March 24th 2016.

- Position converter now checks for selection of gene/transcript (#346).
- Update about page and link to new mailing list (#353).
- Batch jobs processor is more robust on errors (#356).

3.2.20 Version 2.0.16

Released on February 25th 2016.

- Fixed transcript naming in mapping webservices (#147). See below for *details*.
- Support LRG transcripts in the position converter (#147).
- Don't report `ext*?` when variant RNA has stop codon (#146).

The following three webservice methods return a list of transcript identifiers for some query:

- *getTranscriptsRange*

- *getTranscripts*
- *getTranscriptsByGeneName*

Previously they didn't work correctly for LRG transcripts (a bogus version was included and no transcript was selected) and RefSeq transcripts on mtDNA (no transcript was selected).

Additionally, the *getTranscriptsMapping* webservice method didn't return enough information to construct a complete transcript name. This is now reported in a new *transcript* field.

Finally, the *getTranscriptsRange* method now optionally includes version numbers with the boolean *versions* argument (default *false*).

3.2.21 Version 2.0.15

Released on January 6th 2016.

- Speedup NCBI mapview file import (#124).
- Parse genbank file without VERSION field (#126).
- Fix issue where some transcripts would not show in the legend (#136).
- Don't discard complete gene from GenBank file when it has incomplete but also complete features annotated (#138)

3.2.22 Version 2.0.14

Released on November 9th 2015.

- E-mail address is now optional in batch job website interface (#119).
- Use Mailcheck in the batch jobs form (#107).
- Optional email for batch jobs from webservices (#104).
- Process batch jobs grouped by email address (#101).
- Use interval binning scheme on transcript mappings (#100).
- Back translator interface (#74).
- Transcript-protein links are now cached in Redis (#94).
- Added *Ter* as a valid amino acid in the HGVS grammar (#90).
- Refactoring of unit tests (#88, #89).

3.2.23 Version 2.0.13

Released on October 1st 2015.

- Fix query bug in update transcript-protein links (#85).

3.2.24 Version 2.0.12

Released on September 30th 2015.

- Fix off-by-one in slicing chromosome by gene name (#79).

- Document scheme used for all positions and ranges (#79).
- Show diff for variant protein from non-reference start codon (#78).
- Visualise protein change, also with alternative start (#72).
- Translate alternative start to M, also in variant (#72).
- Added Baker's yeast (SacCer_Apr2011/sacCer3) assembly (#73).

3.2.25 Version 2.0.11

Released on August 6th 2015.

- Fix bug in recognizing `p.(=)` (was reported as `p.?`) (#65).

3.2.26 Version 2.0.10

Released on July 21st 2015.

- Don't crash the position converter on transcript mappings containing no exons (#63).
- Use the notation for an uncertain stop codon, e.g., `p.(Gln730Profs*?)` instead of `p.(Gln730Profs*96)` when a variant results in a frame shift or extension and we don't see a new stop codon in the RNA (#57).
- Added Dog (Broad CanFam3.1/canFam3) assembly for position converter (#56).

3.2.27 Version 2.0.9

Released on July 9th 2015.

- Improvements in encoding detection of uploaded batch files (#52, #53).
- Usability improvements in reading DNA for description extractor (#54).

3.2.28 Version 2.0.8

Released on May 27th 2015.

- The HGVS variant description extractor package is available through the Description Extractor interface (#1).
- GitHub project moved from LUMC/mutalyzer to mutalyzer/mutalyzer.

3.2.29 Version 2.0.7

Released on March 27th 2015.

- Return fault code and description on RPC service errors (#31).
- Use esummary 2.0 response format (#32).

3.2.30 Version 2.0.6

Released on February 10th 2015.

- Added *getGeneLocation* webservice method. Given a gene symbol and optional genome build, it returns the location of the gene (#28).
- Discard incomplete genes in genbank reference files (#26).

3.2.31 Version 2.0.5

Released on December 16th 2014.

- New website layout by Landscape (GitLab!26).
- Source code moved from GitLab to GitHub.
- Automated unit tests on Travis CI (#16).
- Developer documentation hosted on Read the Docs (#17).

3.2.32 Version 2.0.4

Released on November 19th 2014.

- Many string encoding related fixes. Summarizing, Mutalyzer should now be completely aware of input and output string encodings, in all interfaces. Internally, all strings are unicode strings. This comes with minor changes in the webservice definitions, which most clients will probably not notice (GitLab!25).
- Don't crash on mail errors in the batch scheduler (GitLab#30).
- Fix importing transcript mappings from UCSC database (GitLab#9).
- Rename GRCh36 to NCBI36 (GitLab#8).
- Updated all Python dependencies to their latest versions.
- Mutalyzer is now Open Source! Source code is available under the AGPL and documentation under the CC-by-sa license.

3.2.33 Version 2.0.3

Released on September 20th 2014.

- Fix several error cases in LOVD2 view on the name checker.

3.2.34 Version 2.0.2

Released on October 9th 2014.

- Fix incorrect GRCm38 chromosome accession number versions.
- Fix crash in position converter batch jobs.
- Upgrade the webservice library we use (Spyne, from 2.10.10 to 2.11.0). This potentially affects behaviour of both our SOAP and HTTP/RPC+JSON webservices, although our tests did not show any problems.

3.2.35 Version 2.0.1

Released on September 27th 2014.

- Fix POST requests to the HTTP/RPC+JSON webservice. This was a regression from version 2.0.beta-33. Thanks to Ken Doig for reporting the issue.

3.2.36 Version 2.0.0

Released on September 26th 2014.

This release does not bring many new features, but comes with significant changes to the technical infrastructure. GitLab!6 tracks most of this.

Some highlights especially users of the webservices should be aware of:

- HTTP/RPC+JSON webservice has changed response format (wrapper object removed). See below for an *example*.
- No more plain HTTP access, only redirects to HTTPS.
- Many website entrypoints have changed URLs and form parameter names (the old ones have HTTP redirects).
- Removed old redirects from paths starting with `/2.0/`.
- In maintenance mode, all requests get a *Service Temporarily Unavailable* response with status code 503.

Other changes:

- Upload a genbank file using the SOAP webservice (*uploadGenBankLocalFile*).
- Do not cleanup the cache during request handling (GitLab#18).
- Add GRCh38 (hg38) assembly (GitLab!20).
- Move from nose to pytest for unit tests (GitLab!23).
- Fix running Mutalyzer in a virtual environment and have an up-to-date `requirements.txt` for pip (GitLab!4).
- Switch from TAL to Jinja2 (GitLab!3).
- Refactor user interfaces (GitLab!5).
- Move from configobj to Python module based config (GitLab!7).
- Use SQLAlchemy as ORM (GitLab!8).
- Use Redis for stat counters (GitLab!10).
- Port website from web.py to Flask (GitLab!11).
- Isolated unit tests using fixtures and an in-memory database (GitLab!12).
- Display announcement on website (GitLab!14).
- Database migrations with Alembic (GitLab!15).
- Update documentation and use Sphinx (GitLab!16).
- Move to semantic versioning, starting with version 2.0.0 (GitLab!22).
- Add 404 not found page.
- Don't auto remove comma characters in syntax checker.
- Add a dash (−) as an allowed character in the gene name.

- Range, reverse complement range, and compound insertions/insertion-deletions.

The wrapper object has been removed from the HTTP/RPC+JSON webservice response format. As an example, consider an old response format for the *checkSyntax* method:

```
{
  "checkSyntaxResponse": {
    "checkSyntaxResult": {
      "valid": true,
      "messages": {
        "SoapMessage": []
      }
    }
  }
}
```

The new response format is:

```
{
  "valid": true,
  "messages": []
}
```

3.2.37 Version 2.0.beta-33

Released on August 19th 2014.

- Link to Upcoming server update announcement.

3.2.38 Version 2.0.beta-32

Released on June 26th 2014.

- Link to Visual interface for Variant Description Extractor announcement.

3.2.39 Version 2.0.beta-31

Released on March 27th 2014.

- Due to incorrect interpretation, temporarily only support one CDS per transcript (ignore all others) in LRG.
- Due to incorrect interpretation, temporarily ignore transcripts without a fixed id.

3.2.40 Version 2.0.beta-30

Released on February 18th 2014.

- Handle NCBI Entrez response validation errors (fixes, among other things, LOVD Trac#29).
- Loosen error severity when CDS cannot be translated.
- Mutalyzer development migrated from Subversion to Git for version control.

3.2.41 Version 2.0.beta-29

Released on October 11th 2013.

- Add Jonathan Vis attribution and COMMIT logo to about page.

3.2.42 Version 2.0.beta-28

Released on September 18th 2013.

- Enable the HTTP/RPC+JSON web service to be used with POST requests.

3.2.43 Version 2.0.beta-27

Released on June 18th 2013.

- Fix caching transcript-protein links from NCBI, reducing impact of NCBI communication problems.

3.2.44 Version 2.0.beta-26

Released on April 9th 2013.

- Added mm10 (Mouse) transcript mappings to position converter.
- LRG parser updated to LRG 1.7 schema (Trac#127).

3.2.45 Version 2.0.beta-25

Released on March 25th 2013.

- Detect incorrect exon annotation in transcript references.
- Move documentation to Trac.
- Exon table is included in *runMutalyzer* webservice results.
- Temporarily disable frameshift detection in experimental description extractor (Trac#124).
- Allow selectors on transcript references in position converter.
- Syntax checker now supports protein level variant descriptions.

3.2.46 Version 2.0.beta-24

Released on December 10th 2012.

- Rename some warning codes (webservice API) (Trac#98).
- Variants on mtDNA in position converter.

3.2.47 Version 2.0.beta-23

Released on November 8th 2012.

No user-visible changes.

3.2.48 Version 2.0.beta-22

Released on November 2nd 2012.

- Submitting batch jobs via the web services (Trac#115).
- Allow for leading whitespace in batch job input (Trac#107).
- New *descriptionExtract* webservice function.
- Name checker now includes description extractor output as an experimental service.
- Slice chromosome by gene name in reference file loader is now case insensitive (Trac#118).
- Warn on missing positioning scheme (Trac#114).

3.2.49 Version 2.0.beta-21

Released on July 23rd 2012.

- Support compound variants in position converter.
- Support non-coding transcripts in position converter (Trac#102).
- Move to new RPC library version, causing slight change in HTTP/RPC+JSON webservice output (more wrappers around output), but fixes Trac#104.
- Fix position converter for delins with explicit deleted sequence.
- Fix description update from Version 2.0.beta-20 to use- notation instead of counting.

3.2.50 Version 2.0.beta-20

Released on July 21st 2012.

- Disabled the `-u` and `+d` convention in favour of the official HGVS recommendations.

3.2.51 Version 2.0.beta-19

Released on June 21st 2012.

- Fix crash on inversions (Trac#99).

3.2.52 Version 2.0.beta-18

Released on June 7th 2012.

- Moved from soaplib to rpclib for webservices (Trac#66).
- Added HTTP/RPC+JSON webservice (Trac#18).
- Fixed name checker errors in some adjacent variants (Trac#83).
- Name checker form now uses GET requests to support easier linking to result pages.
- You can now specify chromosomes by name in the reference file loader (Trac#92).
- Made batch daemon not crash on MySQL restarts (Trac#91).
- Position converter now detects incorrect order in position ranges (Trac#95).

- Added NBIC logo to 'about' page.

3.2.53 Version 2.0.beta-17

Released on April 2nd 2012.

- Fixed crossmapping bug for some transcripts.
- Fixes for NCBI Entrez EFetch Version 2.0 release.
- Better chromosomal variant descriptions.
- Various smaller features and bugfixes.

3.2.54 Version 2.0.beta-16

Released on March 1st 2012.

- Fixed position converter mapping info for some transcripts.
- Fixed deletion with deleted sequence length as argument.

3.2.55 Version 2.0.beta-15

Released on February 20th 2012.

- Added 'Description Extractor' (see the main menu).
- Fixes for NCBI Entrez EFetch Version 2.0 release.
- Added chromosomal positions to *getTranscriptsAndInfo* webservice.
- Fixed chromosome slicing on reverse complement
- Fixed describing NOP variants with =.
- Added Reference sequence info in *runMutalyzer* SOAP function response.
- Fixed mapping info for genes mapped to more than one chromosome.
- Various smaller features and bugfixes.

3.2.56 Version 2.0.beta-14

Released on January 26th 2012.

- Added a SOAP service *getTranscriptsMapping*.
- Various smaller features and bugfixes.

3.2.57 Version 2.0.beta-13

Released on January 25th 2012.

- Accept EX positioning scheme.
- Fix handling of LRG reference sequences.
- Various smaller features and bugfixes.

3.2.58 Version 2.0.beta-12

Released on November 25th 2011.

- Accept plasmid reference sequences.
- View variant position in UCSC Genome Browser (only for transcript references).
- Retry querying dbSNP if it does not respond the first time.
- Support reference GenBank files built from contigs.
- Add optional argument to SOAP service *numberConversion* to map chromosomal locations to any gene.
- Various smaller features and bugfixes.

3.2.59 Version 2.0.beta-11

Released on September 30st 2011.

- Major code refactoring:
  - Mutalyzer is now structured as a proper Python package.
  - Reworked installation and upgrade procedure.
  - Remote installation using Fabric.
  - Batch scheduler is now a proper system daemon.
  - Use mod_wsgi (with web.py) instead of the deprecated mod_python.
  - Added a lot of internal documentation.
  - Introduce unit tests.
  - Handle deletions of entire exons.
  - Added a SOAP service *info*.
  - Handle unknown (fuzzy) intronic positions.
  - Automatic synchronization of database and cache between Mutalyzer installations.
  - Use NCBI instead of UCSC for transcript mapping info.
  - Added a SOAP service *getdbSNPDescriptions*.
  - Moved Trac and Subversion repository to new server.
  - Implement HTTP HEAD method for `/Reference/*` locations.
- Added a SOAP service *ping*.
- Added an optional versions parameter to the SOAP service *getTranscripts*.
- Various smaller features and bugfixes.

3.2.60 Version 2.0.beta-10

Released on July 21st 2011.

- Greatly reduce runtime for large batch jobs.

3.2.61 Version 2.0.beta-9

Released on June 27th 2011.

- Reworked the calculation of new splice site positions.
- Optionally restrict SOAP service *getTranscriptsAndInfo* transcripts to a gene.
- Add raw variants to SOAP service *runMutalyzer* results.
- Provide webservice client examples.
- Various smaller features and bugfixes.

3.2.62 Older versions

The first lines of code for Mutalyzer 2.0 were written July 28th 2009, and version 2.0.beta-8 was released on January 31st 2011. As far as Mutalyzer 1 is concerned, archaeology is not really our field of research.

3.3 Copyright

3.3.1 Authors

Mutalyzer 2 is designed and developed by Jeroen F.J. Laros at the department of Human Genetics at Leiden University Medical Center where it is currently maintained by Martijn Vermaat.

The following people have worked on developing Mutalyzer:

- Jeroen Laros
- Martijn Vermaat
- Jonathan Vis
- Gerben Stouten
- Gerard Schaafsma
- Daria Dvorkina
- Alisa Muraveva

Specifications are given by Peter E.M. Taschner and Johan T. den Dunnen.

Furthermore we would like to thank the following people for their valuable work on previous versions that acted as a guideline for the development of the current version:

- Ernest van Ophuizen
- Martin Wildeman
- Corinne Bareil

CHAPTER 4

Indices and tables

- genindex
- modindex
- search