amaGama Documentation

Release 0.0.0

Translate.org.za

March 24, 2016

Contents

amaGama is a web service implementing a large-scale translation memory. A translation memory is a database of previous translations which can be searched to find good matches to new strings.

amaGama is implemented in Python on top of PostgreSQL. There are currently no releases of the software, but the source code is available at Github.

A public deployment of amaGama is available, providing both a public API and a web search interface on top of the API, and is usable from Virtaal and Pootle.

amaGama is the Zulu word for *words*.

CHAPTER 1

Installing amaGama

Want to try amaGama? This will guide you through installing amaGama and its requirements.

1.1 Dependencies

amaGama requires the following dependencies:

- **Python 2**: 2.6 or later.
- **PostgreSQL**: Tested on 8.3 and 8.4.

There are also some dependencies that we strongly recommend to use, but are optional:

- **git**: Necessary to get amaGama.
- **virtualenv**: Provides an isolated environment or virtualenv.
- **virtualenvwrapper**: To ease handling virtualenvs.

Consult the specifics for your operating system in order to get each above package installed successfully.

1.2 Setting up a virtualenv

The use of virtualenvs allows to install all the requirements at specific versions without interfering with system-wide packages. To create a virtualenv just run:

```
$ mkvirtualenv amagama
```

1.3 Getting amaGama

There is no package for amaGama, so you will need to run it from a git checkout:

```
(amagama) $ git clone https://github.com/translate/amagama.git
(amagama) $ cd amagama
```

1.4 Installing the requirements

Then install the requirements:

```
(amagama) $ pip install -r requirements/recommended.txt
```

After installing the amaGama requirements, you can safely start amaGama installation.

1.5 Creating the database

amaGama requires a PostgreSQL database to store translations. So create an empty database, for example doing the following:

```
$ su root
# su postgres
$ createdb -E UTF-8 amagama
```

Note: You might see an error like:

```
createdb: database creation failed: ERROR: new encoding (UTF8) is
incompatible with the encoding of the template database (SQL_ASCII)
```

This could happen because the database was installed in the *"C"* locale. This might be fixed by doing the following:

```
$ createdb -E UTF-8 -T template0 amagama
```

1.6 Adjusting the settings

The next step is to adjust amaGama settings to include the right database connection configuration, and perhaps change any other setting. Check the *amaGama settings documentation* in order to know how to do it.

Note: One simple change that you should most likely make on a toy installation is to set:

```
DB_HOST = "localhost"
```

This is a side effect of how Postgres is installed on Ubuntu and other systems.

1.7 Making the commands accessible

Since amaGama is not installed we need to make accessible its commands:

```
$ export PATH=$(pwd)/bin:$PATH
$ export PYTHONPATH=$(pwd):$PYTHONPATH
```

1.8 Preparing the database

The first step after editing the settings is to prepare database tables for each source language you will use (you can add more languages later):

```
$ amagama-manage initdb -s en -s fr
```

1.9 Next steps

Now that you have managed to install amaGama you will probably want to know how to:

- *Manage amaGama*
- *Import translations* to amaGama
- *Run amaGama*

CHAPTER 2

amaGama settings

amaGama has some settings that allow to tune how it behaves. Below you can see a detailed description for each setting and its default values.

amaGama settings are stored in `amagama/settings.py`.

2.1 Global settings

Settings to define amaGama server behavior.

DEBUG Default: `False`

Indicates if the debug mode is enabled.

SECRET_KEY Default: `foobar`

Indicates the secret key to use for keeping the sessions secure.

ENABLE_WEB_UI Default: `False`

Indicates if the web interface is enabled.

ENABLE_DATA_ALTERING_API Default: `False`

Indicates if the part of the amaGama API that allows data to be altered is enabled.

This doesn't affect to the part of the API that is used to perform queries that don't alter the data. For example retrieving translations is always enabled.

2.2 Database settings

Settings used for connecting to the amaGama database.

DB_HOST Default: `"localhost"`

Hostname of the server where the amaGama database is located.

DB_NAME Default: `"amagama"`

amaGama database name.

DB_PASSWORD Default: `""`

Password for the amaGama database user.

DB_PORT Default: `"5432"`

Port number where the database server holding the amaGama database is listening.

DB_USER Default: `"postgres"`

User name for connecting to the amaGama database.

2.3 Database pool settings

Settings for the database pool.

DB_MAX_CONNECTIONS Default: `20`

Maximum number of connections that the pool database will handle.

DB_MIN_CONNECTIONS Default: `2`

Number of connections to the database server that are created automatically in the database pool.

2.4 Levenshtein settings

Settings for Levenshtein algorithm. See Levenshtein distance for more information.

MAX_CANDIDATES Default: `5`

The maximum number of results returned. This can be overridden by providing another value using a *query string*.

MAX_LENGTH Default: `1000`

Maximum length of the string. If the string length is higher then it won't be matched neither returned in the results.

MIN_SIMILARITY Default: `70`

The minimum similarity between the string to be searched and the strings to match.

This can be overridden by providing another value using a *query string*, but there is a hardcoded minimum possible value of `30`. If a lower value is provided then `30` will be used.

CHAPTER 3

Managing amaGama

Note: Please make sure that the **amagama-manage** command *is accessible* in order to be able to use it.

amaGama is managed through the **amagama-manage** command. Try running it with no arguments for usage help:

```
$ amagama-manage
```

The **amagama-manage** command exposes several management subcommands, each having it's own `--help` option that displays its usage information:

```
$ amagama-manage SUBCOMMAND --help
```

See below for the available subcommands.

3.1 Available subcommands

These are the available management subcommands for amaGama:

3.1.1 benchmark_tmdb

This subcommand benchmarks the application by querying for all strings in the given file.

Note: For more information please check the help of this subcommand.

3.1.2 build_tmdb

This subcommand is used to import translations into amaGama from bilingual translation files. Please refer to the *importing translations* section for a complete usage example.

3.1.3 deploy_db

This subcommand is used to optimize the database for deployment. It has no options:

```
$ amagama-manage deploy_db
This will permanently alter the database. Continue? [n] y
Succesfully altered the database for deployment.
```

3.1.4 dropdb

This subcommand is used to drop the tables for one or more source languages from the amaGama database:

```
$ amagama-manage dropdb -s fr -s de
This will permanently destroy all data in the configured database. Continue? [n] y
Succesfully dropped the database for 'fr', 'de'.
```

3.1.5 initdb

This subcommand is used to create the tables in the database for one or several source languages. It can be run several times to specify additional source languages. The following example creates the tables for english and french:

```
$ amagama-manage initdb -s en -s fr
Succesfully initialized the database for 'en', 'fr'.
```

3.1.6 tmdb_stats

This subcommand is used to print out some figures about the amaGama database. It has no options:

```
$ amagama-manage tmdb_stats
Complete database (amagama):        400 MB
Complete size of sources_en:        234 MB
Complete size of targets_en:        160 MB
sources_en (table only):    85 MB
targets_en (table only):    66 MB
sources_en  sources_en_text_idx     83 MB
targets_en  targets_en_unique_idx   79 MB
sources_en  sources_en_text_unique_idx      53 MB
targets_en  targets_en_pkey 16 MB
sources_en  sources_en_pkey 13 MB
```

CHAPTER 4

Importing translations

Note: Please make sure that the **amagama-manage** command *is accessible*.

To populate the amaGama database the **amagama-manage** command **build_tmdb** subcommand should be used:

```
$ amagama-manage build_tmdb --verbose -s en -t ar -i foo.po
Importing foo.po
Succesfully imported foo.po
```

This will parse `foo.po`, assuming that source language is *English (en)* and target language is *Arabic (ar)*, and will populate the database accordingly.

The source and target language options only need to be specified if the file does not provide this information. But if source and target language options are specified they will override the languages metadata in the translation file.

All bilingual formats supported by the Translate Toolkit are supported, including **PO**, **TMX** and **XLIFF**.

If a directory is passed to the `-i` option, then its content will be read recursively:

```
$ amagama-manage build_tmdb --verbose -s en -t gl -i translations/
Importing translations/foo.po
Importing translations/bar.po
Succesfully imported translations/
```

CHAPTER 5

Running amaGama

Note: Please make sure that the **amagama** command *is accessible*.

The **amagama** command will try to use the best pure Python WSGI server to launch amaGama server listening on port `8888`.

```
$ amagama
```

After launching the server you can test that amaGama is working by visiting http://localhost:8888/tmserver/en/ar/unit/file which should display a JSON representation of the *Arabic* translations for the *English* **file** word.

Note: For more options check:

```
$ amagama --help
```

CHAPTER 6

Integrating amaGama with Virtaal

Virtaal has a plugin for the public amaGama server since version **0.7** and it is enabled by default.

amaGama implements the same protocol as *tmserver*, and can be used with Virtaal's *remotetm* plugin, or other software that supports this.

In Virtaal go to *Edit* → *Preferences* → *Plugins* → *Translation Memory* → *Configure* to make sure the remote server plugin is enabled and then close Virtaal.

Edit `~/.virtaal/tm.ini` and make sure there is a `remotetm` section that looks like this:

```
[remotetm]
host = localhost
port = 8888
```

Note: If you are going to use a remote amaGama server this setting needs to be changed accordingly.

Run Virtaal again. You should start seeing results from amaGama (they will be marked as coming from *remotetm*).

CHAPTER 7

Developers

7.1 Contributing

We accept code contributions to amaGama, please use Github pull requests for your changes.

7.1.1 Preparations

You will need a local working copy of amaGama, the best way to achieve that is to follow the *installation guidelines*.

7.1.2 Coding style

Please follow the Translate Toolkit Style Guide.

7.1.3 TODO

An incomplete list of possible TODO items:

- Improve web interface
- Custom index config for source languages not supported by default PostgreSQL install
- Keep track of file's mtime to avoid expensive reparses
- Use memcached to cache results
- Use more permanent caching of Levenshtein distances?
- Use PostgreSQL built-in Levenshtein functions?
- Full text search
- Other search methods and options
- Further documenting of API
- Document the commands
- Document how to deploy amaGama using Apache or other web server

7.2 amaGama API

7.2.1 TM suggestion request

The URL structure for requesting TM suggestions is `<SERVER>/tmserver/<SOURCE_LANGUAGE>/<TARGET_LANGUAGE>/un` where:

Placeholder	Description
<SERVER>	The URL of the amaGama server
<SOURCE_LANGUAGE>	Source language code: de, en, en_GB
<TARGET_LANGUAGE>	Target language: ar, es_AR, fr, hi
<QUERY>	The URL escaped string to be queried

Note: `<SOURCE_LANGUAGE>` and `<TARGET_LANGUAGE>` should be language codes in the form of **LANG_COUNTRY** where *LANG* is mandatory. **LANG** should be a language code from ISO 639 and **COUNTRY** a country code from ISO 3166. The following are valid examples: ar, de, en, en_GB, es_AR, fr, gl, hi, tlh,...

For example:

```
http://amagama.locamotion.org/tmserver/en/af/unit/Computer
```

Providing options

It is possible to provide some options in the request URL by using a query string with one or more or the following fields.

Option	Description
min_similarity	The minimum similarity between the string to be searched and the strings to match. See Levenshtein distance. Minimum possible value is 30. Default value is 70.
max_candidates	The maximum number of results. Default value is 5.

For example:

```
http://amagama.locamotion.org/tmserver/en/gl/unit/window?min_similarity=31&max_candidates=500
```

7.2.2 TM suggestion results

The results from a TM suggestion request are provided in JSON format. It is a list containing zero or more results. The results contain the following fields:

Field	Description
source	Matching unit's source language text
target	Matching unit's target language test
quality	A Levenshtein distance measure of quality as percent
rank	?

An example:

```
[
  {
    "source": "Computer",
    "quality": 100.0,
    "target": "Rekenaar",
    "rank": 100.0
```

```
  },
  {
    "source": "Computers",
    "quality": 88.888888888888886,
    "target": "Rekenaars",
    "rank": 100.0
  },
  {
    "source": "&Computer",
    "quality": 88.888888888888886,
    "target": "Rekenaar",
    "rank": 100.0
  },
  {
    "source": "_Computer",
    "quality": 88.888888888888886,
    "target": "_Rekenaar",
    "rank": 100.0
  },
  {
    "source": "My Computer",
    "quality": 72.727272727272734,
    "target": "My Rekenaar",
    "rank": 100.0
  }
]
```

D

E

M

S