Weblate Documentation

Release 1.8

Michal Čihař

November 07, 2013

Contents

Contents:

CHAPTER 1

About Weblate

1.1 Project goals

Web based translation with tight git integration supporting wide range of file formats and makes it easy for translators to contribute.

The translations should be kept within same repository as source code and translation process should closely follow development.

There is no plan in heavy conflict resolution as these should be primarily handled on git side.

1.2 Project name

The project is named as mixture of words web and translate.

1.3 Project website

You can find project website at <http://weblate.org/>, there is also demonstration server at <http://demo.weblate.org/>. This documentation can be browsed on <http://weblate.readthedocs.org/>.

1.4 Authors

This tool was written by Michal Čihař <michal@cihar.com>.

CHAPTER 2

Translators guide

2.1 Weblate basics

2.1.1 Projects structure

Each project can contain various subprojects. The reason for this structure is that all subprojects in a project are expected to have a lot in common. Whenever translation is made in single subproject, it is automatically propagated to others within same project (this is especially useful when translating more version of same project).

2.2 Registration and user profile

2.2.1 Registration

While everybody can browse projects, view translations or suggest them, only registered users are allowed to actually save changes and are credited for every translation made.

You can register following two simple steps:

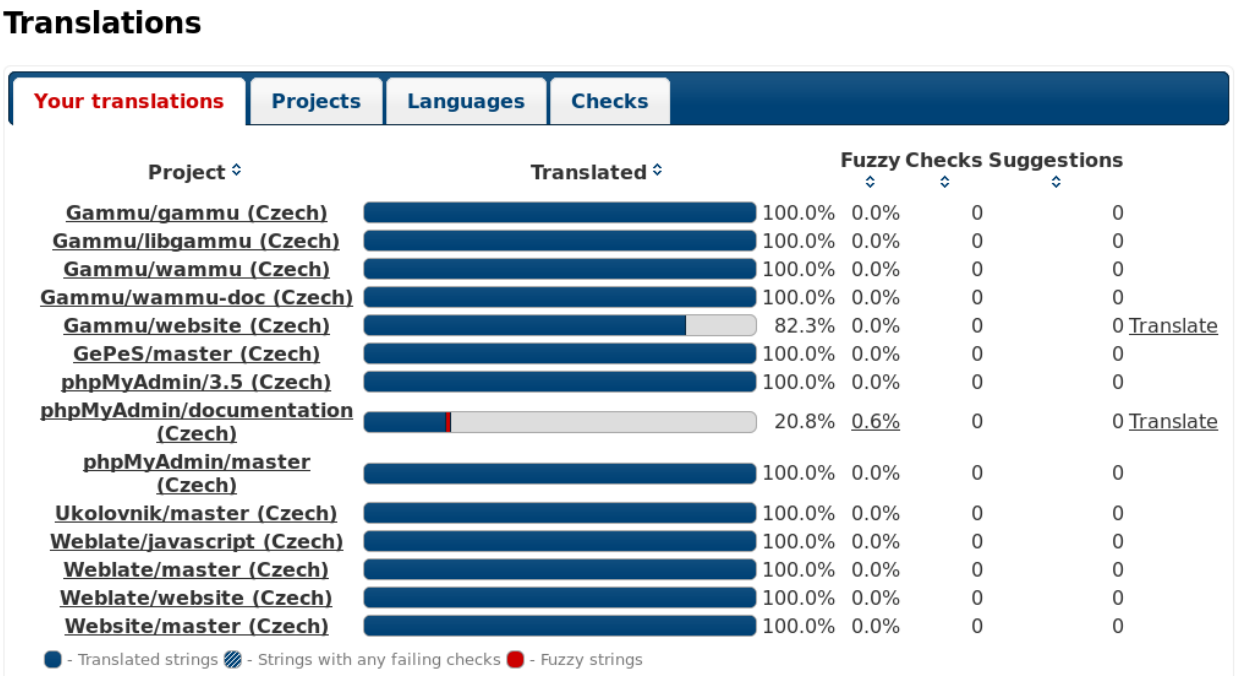
1. Fill out the registration form with your credentials
2. Activate registration by following in email you receive
3. Possibly adjust your profile to choose which languages you know

2.2.2 User profile

User profile contains your preferences, name and email. Name and email are being used in Git commits, so keep this information accurate.

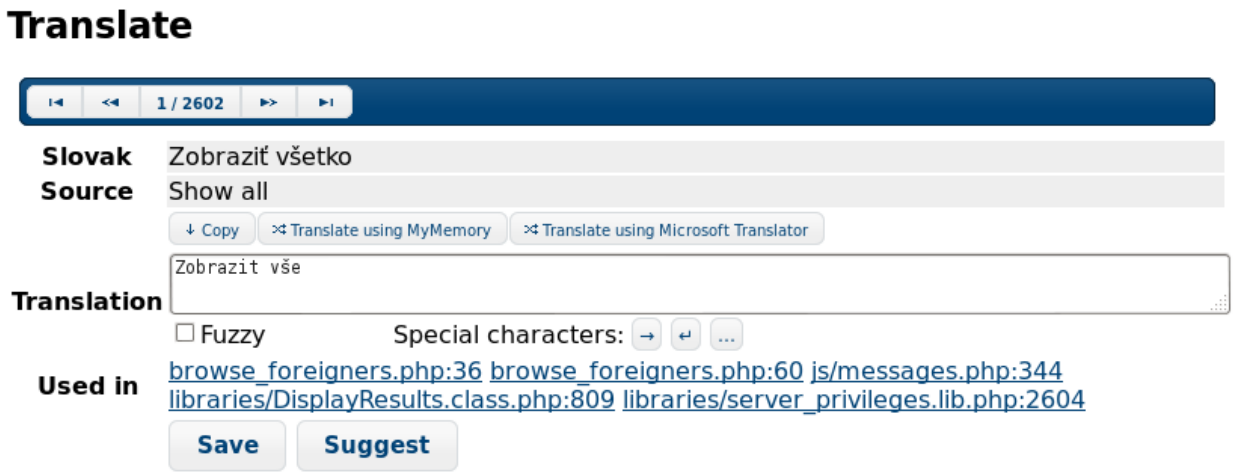
Languages

Choose here which languages you prefer to translate. These will be offered to you on main page to have easier access to translations.



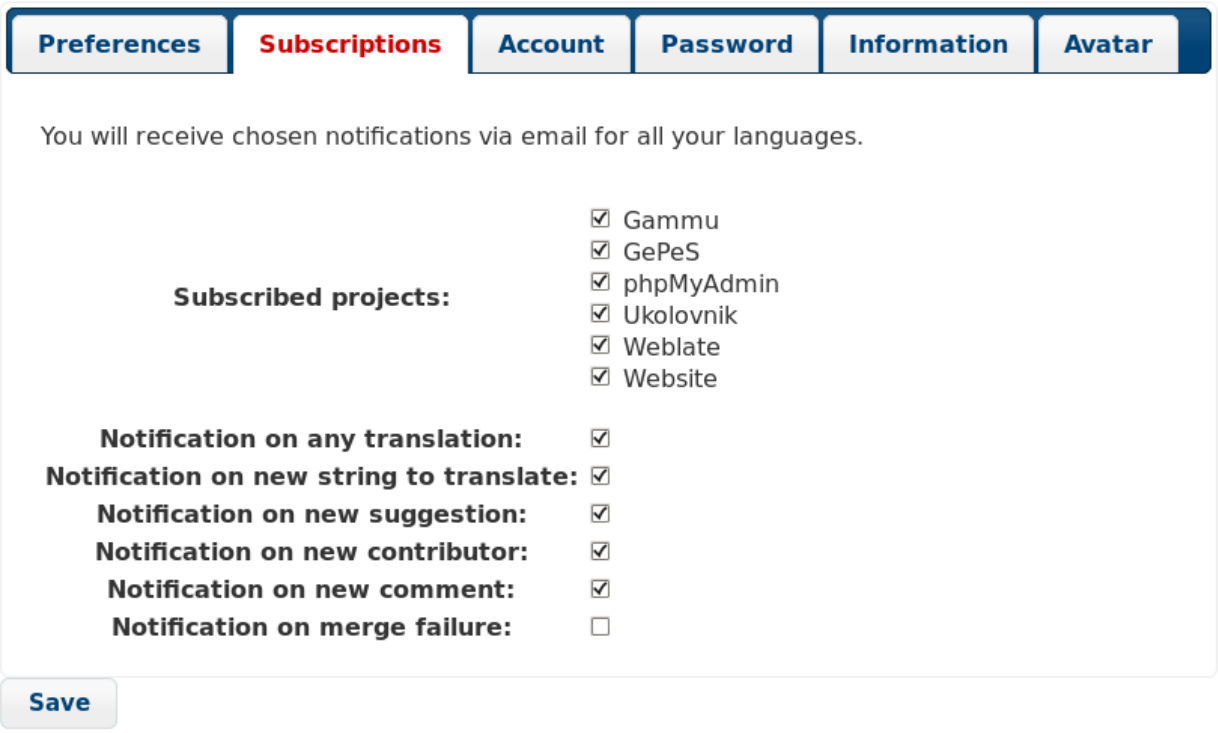
Secondary languages

You can define secondary languages, which will be shown you while translating together with source language. Example can be seen on following image, where Czech language is shown as secondary:



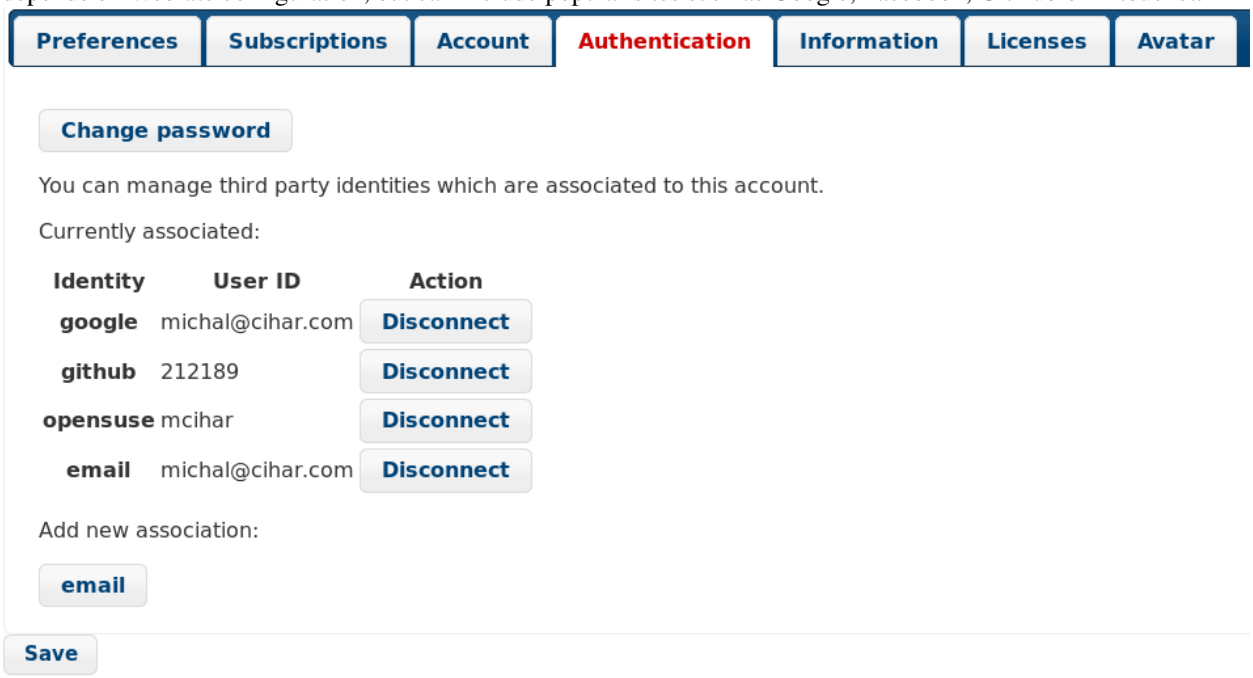
Subscriptions

You can subscribe to various notifications on *Subscriptions* tab. You will receive notifications for selected events on chosen projects for languages you have indicated for translation (see above).



Authentication

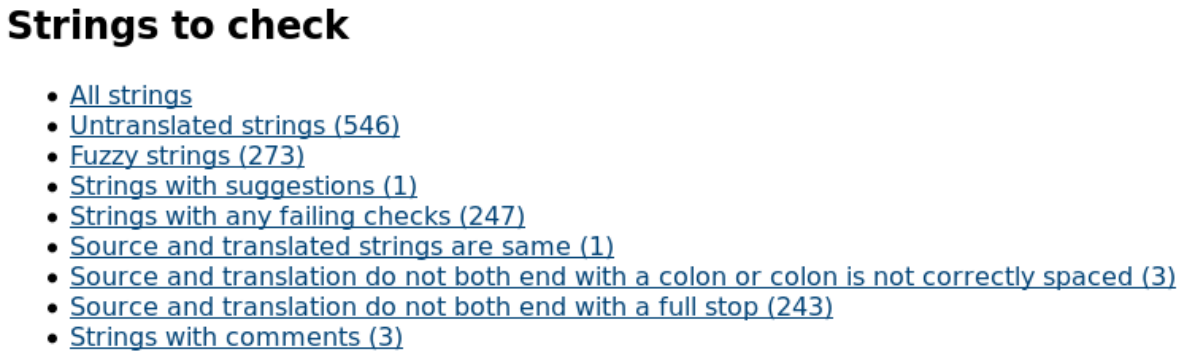
On the *Authentication* tab you can connect various services which you can use to login into Weblate. List of services depends on Weblate configuration, but can include popular sites such as Google, Facebook, GitHub or Bitbucket.



2.3 Translating using Weblate

2.3.1 Translation links

Once you navigate to a translation, you will be shown set of links which lead to translation. These are results of various checks, like untranslated or fuzzy strings. Should no other checks fire, there will be still link to all translations. Alternatively you can use search field to find translation you need to fix.



2.3.2 Suggestions

As an anonymous user, you have no other choice than making a suggestion. However if you are logged in you can still decide to make only a suggestion instead of saving translation, for example in case you are unsure about the translation and you want somebody else to review it.

Note: Permissions might vary depending on your setup, what is described is default Weblate behaviour.

2.3.3 Translating

On translate page, you are shown source string and edit area for translating. Should the translation be plural, multiple source strings and edit areas are shown, each described with label for plural form.

Any special whitespace chars are underlined in red and indicated with grey symbols. Also more than one space is underlined in red to allow translator to keep formatting.

There are various extra information which can be shown on this page. Most of them are coming from the project source code (like context, comments or where the message is being used). When you configure secondary languages in your preferences, translation to these languages will be shown.

Bellow translation can be also shown suggestions from other users, which you can accept or delete.

Translation context

Translation context part allows you to see related information about current string.

Nearby messages Displays messages which are located nearby in translation file. These usually are also used in similar context and you might want to check them to keep translation consistent.

Similar messages Messages which are similar to currently one, which again can help you to stay consistent within translation.

All locations In case message appears in multiple places (eg. multiple subprojects), this tab shows all of them and for inconsistent translations (see *Inconsistent*) you can choose which one to use.

Glossary Displays words from project glossary which are used in current message.

Recent edits List of people who have changed this message recently using Weblate.

Project Project information like instructions for translators or information about Git repository.

If translation format supports it, you can also follow links to source code which contains translated strings.

Translation history

Every change is by default (unless disabled in subproject settings) saved in the database and can be reverted. Of course you can still also revert anything in underlaying version control system.

2.3.4 Export and import

Weblate supports both export and import of translation files. This allows you to work offline and then merge changes back. Your changes will be merged within existing translation (even if it has been changed meanwhile).

Note: This ability might be limited by *Access control*.

Import method

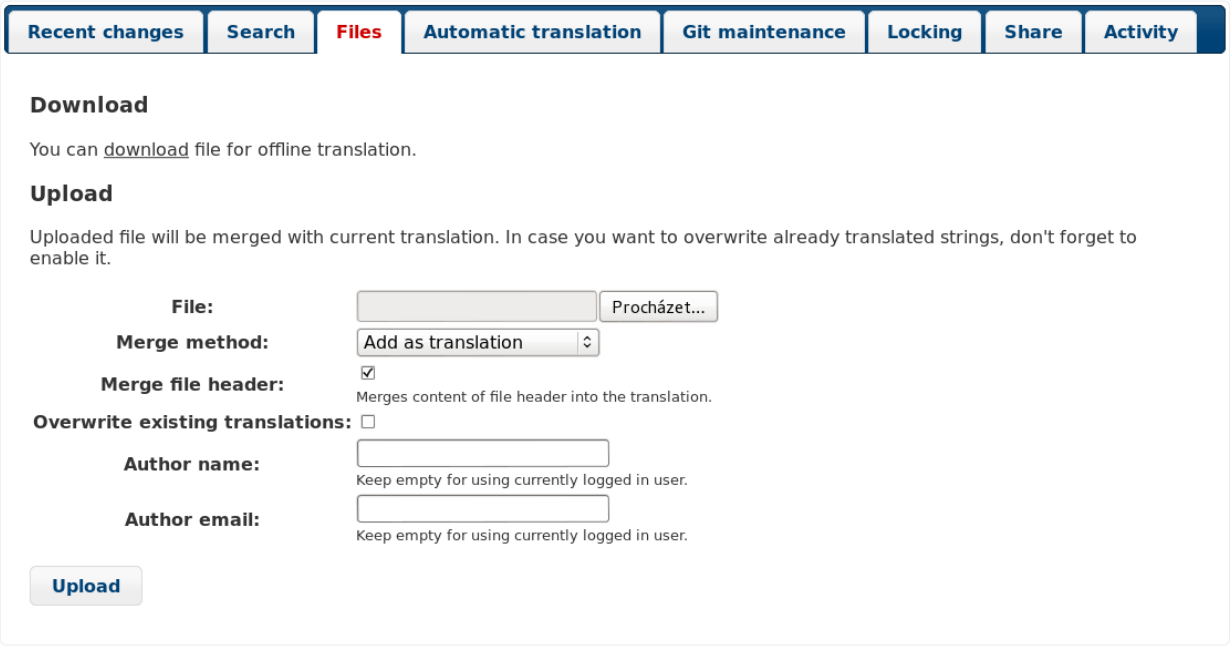
You can choose how imported strings will be merged out of following options:

Add as translation Imported translations are added as translation. This is most usual and default behavior.

Add as a suggestion Imported translations are added as suggestions, do this when you want to review imported strings.

Add as fuzzy translation Imported translations are added as fuzzy translations. This can be useful for review as well.

Additionally, when adding as a translation, you can choose whether to overwrite already translated strings or not.



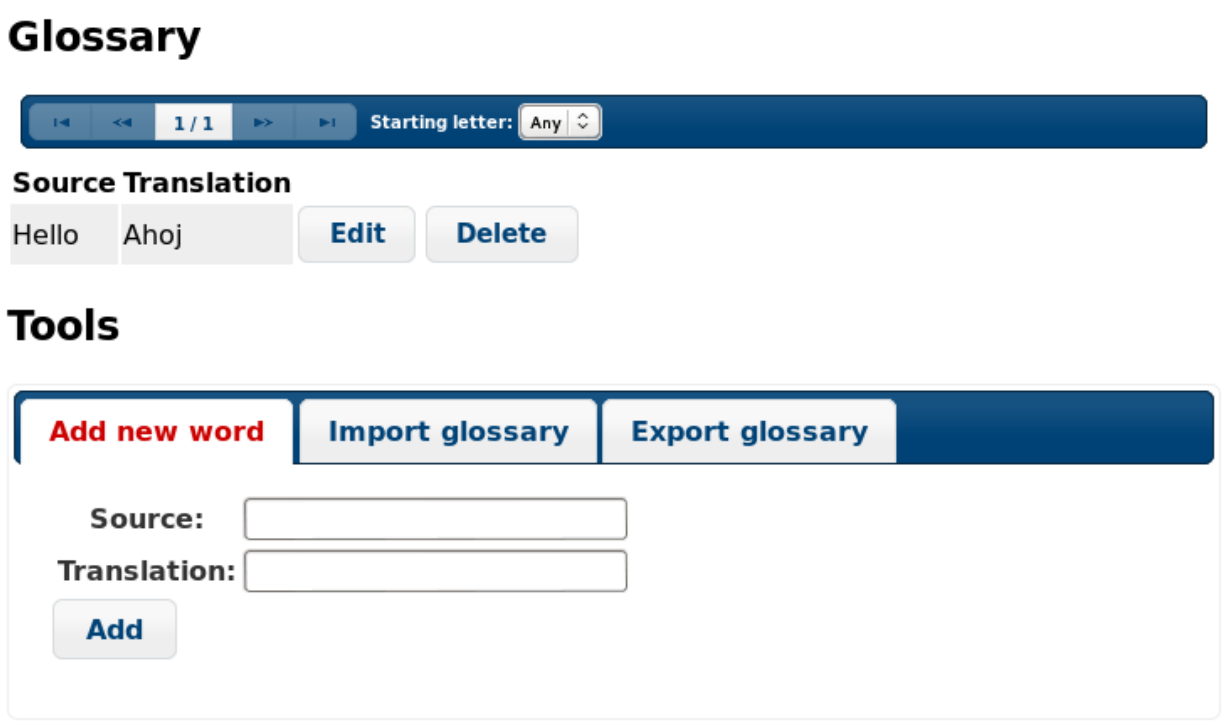
2.3.5 Glossary

Each project can have assigned glossary for any language. This could be used for storing terminology for given project, so that translations are consistent. You can display terms from currently translated string in bottom tabs.

Managing glossaries

On project page, on *Glossaries* tab, you can find link *Manage all glossaries*, where you can start new glossaries or edit existing ones. Once glossary is existing, it will also show up on this tab.

On further page, you can choose which glossary to manage (all languages used in current project are shown). Following this language link will lead you to page, which can be used to edit, import or export the glossary:



2.3.6 Machine translation

Based on configuration and your language, Weblate provides buttons for following machine translation tools.

All machine translations are available on single tab on translation page.

See Also:

Machine translation setup

2.4 Checks and fixups

2.4.1 Automatic fixups

In addition to *Quality checks*, Weblate can also automatically fix some common errors in translated strings. This can be quite powerful feature to prevent common mistakes in translations, however use it with caution as it can cause silent corruption as well.

See Also:

`AUTOFIX_LIST`

2.4.2 Quality checks

Weblate does wide range of quality checks on messages. The following section describes them in more detail. The checks take account also special rules for different languages, so if you think the result is wrong, please report a bug.

See Also:

CHECK_LIST

Translation checks

These are executed on every translation change and help translators to keep good quality of translations.

Not translated

The source and translated strings are the same at least in one of the plural forms. This check ignores some strings which are quite usually same in all languages and strips various markup, which can occur in the string, to reduce number of false positives.

Starting newline

Source and translated do not both start with a newline.

Trailing newline

Source and translated do not both end with a newline.

Starting spaces

Source and translation do not both start with same number of spaces. Space in beginning is usually used for indentation in the interface and thus is important.

Trailing space

Source and translated do not both end with a space.

Trailing stop

Source and translated do not both end with a full stop. Full stop is also checked in various language variants (Chinese, Japanese, Devanagari or Urdu).

Trailing colon

Source and translated do not both end with a colon or the colon is not correctly spaced. This includes spacing rules for French or Breton. Colon is also checked in various language variants (Chinese or Japanese).

Trailing question

Source and translated do not both end with a question mark or it is not correctly spaced. This includes spacing rules for French or Breton. Question mark is also checked in various language variants (Armenian, Arabic, Chinese, Korean, Japanese, Ethiopic, Vai or Coptic).

Trailing exclamation

Source and translated do not both end with an exclamation mark or it is not correctly spaced. This includes spacing rules for French or Breton. Exclamation mark is also checked in various language variants (Chinese, Japanese, Korean, Armenian, Limbu, Myanmar or Nko).

Trailing ellipsis

Source and translation do not both end with an ellipsis. This only checks for real ellipsis (*u2026*) not for three commas (...).

See Also:

https://en.wikipedia.org/wiki/Ellipsis

Python format

Python format string does not match source.

See Also:

http://docs.python.org/2.7/library/stdtypes.html#string-formatting

Python brace format

Python brace format string does not match source.

See Also:

http://docs.python.org/3.3/library/string.html#string-formatting

PHP format

PHP format string does not match source.

See Also:

http://www.php.net/manual/en/function.sprintf.php

C format

C format string does not match source.

See Also:

https://en.wikipedia.org/wiki/Printf_format_string

Missing plurals

Some plural forms are not translated. Check plural form definition to see for which counts each plural form is being used.

Inconsistent

More different translations of one string in a project. This can also lead to inconsistencies in displayed checks. You can find other translations of this string on *All locations* tab.

Mismatched \n

Number of \n in translation does not match source. Usually escaped newlines are important for formatting program output, so this should match to source.

Mismatched BBcode

BBcode in translation does not match source. The method for detecting BBcode is currently quite simple.

Zero-width space

Translation contains extra zero-width space (<U+200B>) character. This character is usually inserted by mistake.

See Also:

https://en.wikipedia.org/wiki/Zero-width_space

XML tags mismatch

XML tags in translation do not match source. This usually means resulting output will look different. In most cases this is not desired result from translation, but occasionally it is desired.

Source checks

Source checks can help developers to improve quality of source strings.

Optional plural

The string is optionally used as plural, but not using plural forms. In case your translation system supports this, you should use plural aware variant of it.

For example with Gettext in Python it could be:

```
from gettext import ngettext

print ngettext('Selected %d file', 'Selected %d files', files) % files
```

Ellipsis

The string uses three dots (...) instead of an ellipsis character (…). Using Unicode character is in most cases better approach and looks better.

See Also:

https://en.wikipedia.org/wiki/Ellipsis

Multiple failing checks

More translations of this string have some failed quality checks. This is usually indication that something could be done about improving the source string.

This check can be quite often caused by missing full stop at the end of sentence or similar minor issues which translators tend to fix in translations.

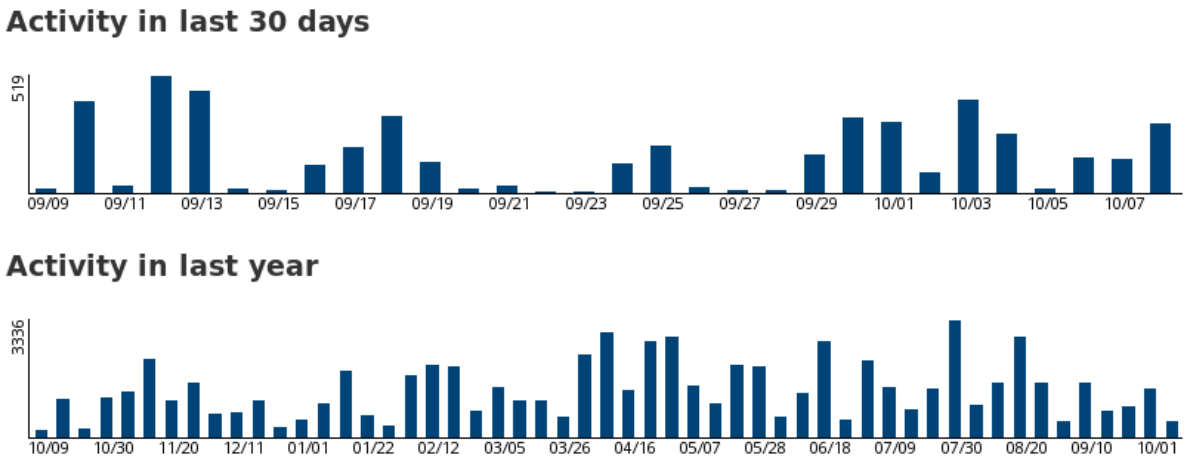
CHAPTER 3

Application developer guide

Using Weblate for translating your projects can bring you quite a lot of benefits. It's only up to you how much of that you will use.

3.1 Activity reports

You can check activity reports for translations, project or individual users.



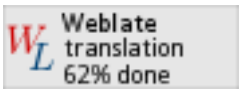
3.2 Promotion the translation

Weblate provides you widgets to share on your website or other sources to promote translation project. It also has nice welcome page for new contributors to give them basic information about the translation. Additionally you can share information about translation using Facebook or Twitter. All these possibilities can be found on *Share* tab. Example of status badges for Weblate itself are show below.

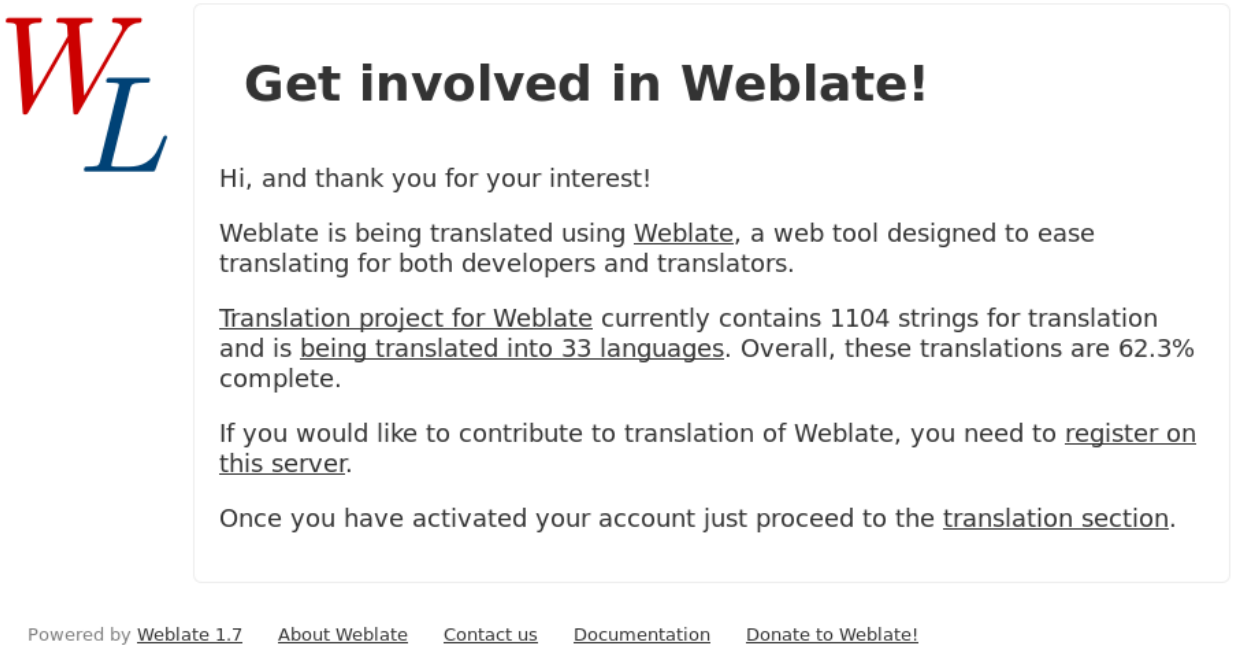
Small badge often used to quickly see status of a project: 

Big badge with status details useful for

inclusion on a web page: 

Small badge with status useful for inclusion on a web page: 

All these badges come with links to simple page which explains users how to translate using Weblate:



Get involved in Weblate!

Hi, and thank you for your interest!

Weblate is being translated using Weblate, a web tool designed to ease translating for both developers and translators.

Translation project for Weblate currently contains 1104 strings for translation and is being translated into 33 languages. Overall, these translations are 62.3% complete.

If you would like to contribute to translation of Weblate, you need to register on this server.

Once you have activated your account just proceed to the translation section.

Powered by Weblate 1.7 About Weblate Contact us Documentation Donate to Weblate!

3.3 Reviewing source strings

3.3.1 Source strings checks

Weblate includes quite a lot of *Quality checks*. Some of them also focus on quality of source strings. These can give you some hints for making strings easier to translate. You can check failing source checks on *Source* tab of every subproject.

3.3.2 Failing checks on translation

On the other side, failing translation checks might also indicate problem in the source strings. Translators often tend to fix some mistakes in translation instead of reporting it - typical example is missing full stop at the end of sentence, but there are more such cases.

Reviewing all failing checks on your translation can bring you valuable feedback for improving source strings as well.

3.3.3 String comments

Weblate allows translators to comment on both translation and source strings. Each *Subproject* can be configured to receive such comments on email address and sending this to developers mailing list is usually best approach. This way you can monitor when translators find problems and fix them quickly.

CHAPTER 4

Administrators guide

4.1 Quick setup guide

Note: This is just a quick guide for installing and starting to use Weblate for testing purposes. Please check *Installation instructions* for more real world setup instructions.

4.1.1 Installing from sources

1. Install all required dependencies, see *Requirements*.
2. Grab Weblate sources (either using Git or download a tarball) and unpack them.
3. Copy `weblate/settings_example.py` to `weblate/settings.py` and adjust it to match your setup. You will at least need to configure database connection (possibly adding user and creating the database). Check *Configuration* for Weblate specific configuration options.
4. Build Django tables and initial data:

```
./manage.py syncdb
./manage.py migrate
./scripts/generate-locales # If you are using Git checkout
```

5. Configure webserver to serve Weblate, see *Running server*.

4.1.2 Using prebuilt appliance

1. Download the appliance and start it. You need to choose format depending on your target environment.
2. Everything should be set up immediately after boot, though you will want to adjust some settings to improve security, see *Appliance*.

4.1.3 Adding translation

1. Open admin interface (http://example.org/admin/) and create project you want to translate. See *Project* for more details.

All you need to specify here is project name and it's website.

2. Create subproject which is the real resource for translating - it points to Git repository and selects which files to translate. See *Subproject* for more details.

   The important fields here being subproject name, Git repository address and mask for finding translatable files. Weblate supports wide range of formats including Gettext PO files, Android resource strings, OS X string properties, Java properties or Qt Linguist files, see *Supported formats* for more details.

3. Once above is completed (it can be lengthy process depending on size of your Git repository and number of messages to translate), you can start translating.

4.2 Installation instructions

4.2.1 Requirements

Python (2.7) http://www.python.org/

Django (>= 1.4) https://www.djangoproject.com/

Translate-toolkit (>= 1.9.0, 1.10.0 or newer strongly recommended) http://toolkit.translatehouse.org/

GitPython (>= 0.3.2) https://github.com/gitpython-developers/GitPython

Git (>= 1.7.2) http://git-scm.com/

python-social-auth (>= 0.1) http://psa.matiasaguirre.net/

Whoosh (>= 2.5, 2.5.2 is recommended) http://bitbucket.org/mchaput/whoosh/

PIL or Pillow library http://python-imaging.github.io/

south http://south.aeracode.org/

libravatar (optional for federated avatar support) https://pypi.python.org/pypi/pyLibravatar

PyICU (optional for proper sorting of strings) https://pypi.python.org/pypi/PyICU

Database backend Any database supported in Django will work, check their documentation for more details.

Requirements on Debian or Ubuntu

On Debian or Ubuntu, most of requirements are already packaged, to install them you can use apt-get:

```
apt-get install python-django translate-toolkit python-git \
    python-whoosh python-pil python-django-south python-libravatar python-pyicu

# Optional for database backend

apt-get install python-mysqldb   # For MySQL
apt-get install python-psycopg2  # For PostgreSQL
```

There is one library not available in Debian so far, so it is recommended to install it using pip:

```
pip install python-social-auth
```

Requirements on openSUSE

All requirements are available either directly in openSUSE or in `devel:languages:python` repository:

```
zypper install python-django python-social-auth translate-toolkit python-GitPython \
    python-whoosh python-imaging python-South
```

Requirements on OSX

If your python was not installed using brew, make sure you have this in your .bash_profile file or executed somehow:

```
export PYTHONPATH="/usr/local/lib/python2.7/site-packages:$PYTHONPATH"
```

This configuration make available the installed libraries to python

Requirements using pip installer

Most requirements can be also installed using pip installer:

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

Also you will need header files for `libxml2` and `libxslt` to compile some of the required Python modules.

On Debian or Ubuntu you can install them using:

```
apt-get install libxml2-dev libxslt-dev
```

On openSUSE or SLES you can install them using:

```
zypper install libxslt-devel libxml2-devel
```

4.2.2 Filesystem permissions

Weblate process needs to be able to read and write to two directories where it keeps data. The `GIT_ROOT` is used for storing Git repositories and `WHOOSH_INDEX` is used for fulltext search data.

The default configuration places them in same tree as Weblate sources, however you might prefer to move these to better location such as `/var/lib/weblate`.

Weblate tries to create these directories automatically, but it will fail when it does not have permissions to do so.

You should also take care when running *Management commands*, as they should be run under same user as Weblate itself is running, otherwise permissions on some files might be wrong.

4.2.3 Installation

Copy `weblate/settings_example.py` to `weblate/settings.py` and adjust it to match your setup. You will probably want to adjust following options:

`ADMINS`

List of site administrators to receive notifications when something goes wrong, for example notifications on failed merge or Django errors.

See Also:

https://docs.djangoproject.com/en/1.4/ref/settings/#admins

ALLOWED_HOSTS

If you are running Django 1.5 or newer, you need to set this to list of hosts your site is supposed to serve. For example:

```
ALLOWED_HOSTS = ['demo.weblate.org']
```

See Also:

https://docs.djangoproject.com/en/dev/ref/settings/#std:setting-ALLOWED_HOSTS

DATABASES

Connectivity to database server, please check Django's documentation for more details.

Note: When using MySQL, don't forget to create database with UTF-8 encoding:

```
CREATE DATABASE <dbname> CHARACTER SET utf8;
```

See Also:

https://docs.djangoproject.com/en/1.4/ref/settings/#databases, https://docs.djangoproject.com/en/1.4/ref/databases/

DEBUG

Disable this for production server. With debug mode enabled, Django will show backtraces in case of error to users, when you disable it, errors will go by email to ADMINS (see above).

Debug mode also slows down Weblate as Django stores much more information internally in this case.

See Also:

https://docs.djangoproject.com/en/1.4/ref/settings/#debug

DEFAULT_FROM_EMAIL

Email sender address for outgoing email, for example registration emails.

See Also:

DEFAULT_FROM_EMAIL documentation

SECRET_KEY

Key used by Django to sign some information in cookies, see *Django secret key* for more information.

SERVER_EMAIL

Email used as sender address for sending emails to administrator, for example notifications on failed merge.

See Also:

SERVER_EMAIL documentation

After your configuration is ready, you can run `./manage.py syncdb` and `./manage.py migrate` to create database structure. Now you should be able to create translation projects using admin interface.

In case you want to run installation non interactively, you can use `./manage.py syncdb --noinput` and then create admin user using `createadmin` command.

You should also login to admin interface (on `/admin/` URL) and adjust default site name to match your domain.

Note: If you are running version from Git, you should also regenerate locale files every time you are upgrading. You can do this by invoking script `./scripts/generate-locales`.

See Also:

Configuration, *Access control*, *Why does registration contain example.com as domain?*

4.2.4 Production setup

For production setup you should do following adjustments:

Disable debug mode

Disable Django's debug mode by:

```
DEBUG = False
```

With debug mode Django stores all executed queries and shows users backtraces of errors what is not desired in production setup.

See Also:

Installation

Properly configure admins

Set correct admin addresses to `ADMINS` setting for defining who will receive mail in case something goes wrong on the server, for example:

```
ADMINS = (
    ('Your Name', 'your_email@example.com'),
)
```

See Also:

Installation

Set correct site name

Adjust site name in admin interface, otherwise links in RSS or registration emails will not work.

See Also:

Why does registration contain example.com as domain?

Enable indexing offloading

Enable `OFFLOAD_INDEXING` to prevent locking issues and improve performance. Don't forget to schedule indexing in background job to keep the index up to date.

See Also:

Fulltext search, `OFFLOAD_INDEXING`

Use powerful database engine

Use powerful database engine (SQLite is usually not good enough for production environment), for example setup for MySQL:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'weblate',
        'USER': 'weblate',
        'PASSWORD': 'weblate',
        'HOST': '127.0.0.1',
        'PORT': '',
    }
}
```

See Also:

Installation, Django's databases

Enable caching

If possible, use memcache from Django by adjusting `CACHES` configuration variable, for example:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

See Also:

Django's cache framework

Configure email addresses

Weblate needs to send out emails on several occasions and these emails should have correct sender address, please configure `SERVER_EMAIL` and `DEFAULT_FROM_EMAIL` to match your environment, for example:

```
SERVER_EMAIL = 'admin@example.org'
DEFAULT_FROM_EMAIL = 'weblate@example.org'
```

See Also:

Installation, DEFAULT_FROM_EMAIL documentation, SERVER_EMAIL documentation

Allowed hosts setup

Django 1.5 and newer require `ALLOWED_HOSTS` to hold list of domain names your site is allowed to serve, having it empty will block any request.

See Also:

https://docs.djangoproject.com/en/dev/ref/settings/#std:setting-ALLOWED_HOSTS

Federated avatar support

By default, Weblate relies on <https://www.libravatar.org/> for avatars. When you install pyLibavatar, you will get proper support for federated avatars.

PyICU library

PyICU library is optionally used by Weblate to sort Unicode strings. This way language names are properly sorted even in non-ASCII languages like Japanese, Chinese or Arabic or for languages with accented letters.

Django secret key

The `SECRET_KEY` setting is used by Django to sign cookies and you should really use own value rather than using the one coming from example setup.

You can generate new key using `examples/generate-secret-key` shipped with Weblate.

See Also:

https://docs.djangoproject.com/en/dev/ref/settings/#std:setting-SECRET_KEY

Admin static files

If you see purely designed admin interface, the CSS files required for it are not loaded. This is usually if you are running in non-debug mode and have not configured your web server to serve them. Recommended setup is described in the *Running server* chapter.

See Also:

Running server

Home directory

The home directory for user which is running Weblate should be existing and writable by this user. This is especially needed if you want to use SSH to access private repositories.

Note: On Linux and other UNIX like systems, the path to user's home directory is defined in `/etc/passwd`. Many distributions default to non writable directory for users used for serving web content (such as `apache`, `www-data` or `wwwrun`, so you either have to run Weblate under different user or change this setting.

See Also:

Private repositories

4.2.5 Running server

Running Weblate is not different from running any other Django based application.

It is recommended to serve static files directly by your web server, you should use that for following paths:

`/media` Serves `media` directory from Weblate.

`/static/admin` Serves media files for Django admin interface (eg. `/usr/share/pyshared/django/contrib/admin/me`

Additionally you should setup rewrite rule to serve `media/favicon.ico` as `favicon.ico`.

See Also:

https://docs.djangoproject.com/en/1.4/howto/deployment/

Sample configuration for Lighttpd

The configuration for Lighttpd web server might look like following (available as `examples/lighttpd.conf`):

```
fastcgi.server = (
    "/weblate.fcgi" => (
        "main" => (
            "socket" => "/var/run/django/weblate.socket",
            "check-local" => "disable",
        )
    ),
)
alias.url = (
    "/media" => "/var/lib/django/weblate/weblate/media/",
    "/static/admin" => "/usr/share/pyshared/django/contrib/admin/static/admin/",
)

url.rewrite-once = (
    "^(/*media.*)$" => "$1",
    "^(/*static.*)$" => "$1",
    "^/*favicon\.ico$" => "/media/favicon.ico",
    "^/*robots\.txt$" => "/media/robots.txt",
    "^(/.*)$" => "/weblate.fcgi$1",
)

expire.url                  = (
    "/media/" => "access 1 months",
    "/static/" => "access 1 months",
    "/favicon.ico" => "access 1 months",
)
```

Sample configuration for Apache

Following configuration runs Weblate as WSGI, you need to have enabled mod_wsgi (available as `examples/apache.conf`):

```
#
# VirtualHost for weblate
#
WSGIPythonPath /usr/share/weblate
<VirtualHost *:80>
    ServerAdmin admin@image.weblate.org
    ServerName image.weblate.org

    DocumentRoot /usr/share/weblate/weblate/media/

    Alias /robots.txt /usr/share/weblate/weblate/media/robots.txt
    Alias /favicon.ico /usr/share/weblate/weblate/media/favicon.ico

    Alias /media/ /usr/share/weblate/weblate/media/
    Alias /doc/ /usr/share/doc/packages/weblate/html/
```

```
    Alias /static/admin /usr/lib/python2.7/site-packages/django/contrib/admin/static/admin/

    <Directory /usr/lib/python2.7/site-packages/django/contrib/admin/static/admin/>
        Order deny,allow
        Allow from all
    </Directory>

    <Directory /usr/share/weblate/weblate/media/>
        Order deny,allow
        Allow from all
    </Directory>

    <Directory /usr/share/doc/packages/weblate/html/>
        Order deny,allow
        Allow from all
    </Directory>

    <Directory /usr/share/weblate/weblate/examples/>
        Order deny,allow
        Allow from all
    </Directory>

    WSGIScriptAlias / /usr/share/weblate/weblate/wsgi.py
    WSGIPassAuthorization On

    <Directory /usr/share/weblate/weblate>
        <Files wsgi.py>
        Order deny,allow
        Allow from all
        </Files>
    </Directory>

</VirtualHost>
```

Sample configuration for nginx

Following configuration runs Weblate as uwsgi under nginx webserver.

Configuration for nginx (also available as `examples/weblate.nginx.conf`):

```
server {
        listen 80;
        server_name weblate;
        root /path/to/weblate/weblate;

        location /favicon.ico {
                root /path/to/weblate/weblate/media;
                expires 30d;
        }

        location /media/ {
                root /path/to/weblate/weblate;
                expires 30d;
        }

        location /robots.txt {
                root /path/to/weblate/weblate/media;
```

```
                expires 30d;
        }

        location /static/admin/ {
                root /usr/local/lib/python2.7/dist-packages/django/contrib/admin;
                expires 30d;
        }

        location / {
                include uwsgi_params;
                uwsgi_pass 127.0.0.1:8080;
        }
}
```

Configuration for uwsgi (also available as `examples/weblate.uwsgi.ini`):

```
[uwsgi]
plugins        = python
master         = true
protocol       = uwsgi
socket         = 127.0.0.1:8080
wsgi-file      = /path/to/weblate/weblate/wsgi.py
python-path    = /path/to/weblate
```

Running Weblate under path

Minimalistic configuration to serve Weblate under `/weblate` (you will need to include portions of above full configuration to allow access to the files). Again using mod_wsgi (also available as `examples/apache-path.conf`):

```
# Example Apache configuration for running Weblate under /weblate path

# Path to Weblate code
WSGIPythonPath /usr/share/weblate

# Path to Weblate WSGI handler
WSGIScriptAlias /weblate "/usr/share/weblate/weblate/wsgi.py"

# Aliases to serve media and static files
Alias /weblate/media/ /usr/share/weblate/weblate/media/
Alias /static/admin /usr/lib/python2.7/site-packages/django/contrib/admin/static/admin/
```

Additionally you will have to adjust `weblate/settings.py`:

```
URL_PREFIX = '/weblate'
```

Note: This is supported since Weblate 1.3.

4.2.6 Appliance

Weblate appliance provides preconfigured Weblate running with MySQL database as backend and Apache as web server. It is provided in many formats suitable for any form of virtualization, cloud or hardware installation.

It comes with standard set of passwords you will want to change:

Username	Password	Scope	Description
root	linux	System	Administrator account, use for local or SSH login
root		MySQL	MySQL administrator
weblate	weblate	MySQL	Account in MySQL database for storing Weblate data
admin	admin	Weblate	Weblate/Django admin user

The appliance is built using SUSE Studio and is based on openSUSE 12.3.

You should also adjust some settings to match your environment, namely:

- *Disable debug mode*
- *Set correct site name*
- *Configure email addresses*

4.2.7 Migrating Weblate to another server

Migrating Weblate to another server should be pretty easy, however it stores data in few locations which you should migrate carefully. The best approach is to stop migrated Weblate for the migration.

Migrating database

Depending on your database backend, you might have several options to migrate the database. The most straightforward one is to dump the database on one server and import it on the new one. Alternatively you can use replication in case your database supports it.

Migrating Git repositories

The Git repositories stored under `GIT_ROOT` need to be migrated as well. You can simply copy them or use **rsync** to do the migration more effectively.

Migrating fulltext index

For the fulltext index (stored in `WHOOSH_INDEX`) it is better not to migrate it, but rather to generate fresh one using `rebuild_index`.

Other notes

Don't forget to move other services which Weblate might have been using like memcached, cron jobs or custom authentication backends.

4.3 Upgrading Weblate

4.3.1 Upgrading

Generic upgrade instructions

Changed in version 1.2: Since version 1.2 the migration is done using South module, to upgrade to 1.2, please see *Version specific instructions*. Before upgrading, please check current *Requirements* as they might have changed.

To upgrade database structure, you should run following commands:

```
./manage.py syncdb
./manage.py migrate
```

To upgrade default set of privileges definitions (optional), run:

```
./manage.py setupgroups
```

To upgrade default set of language definitions (optional), run:

```
./manage.py setuplang
```

Version specific instructions

Upgrade from 0.5 to 0.6

On upgrade to version 0.6 you should run `./manage.py syncdb` and `./manage.py setupgroups --move` to setup access control as described in installation section.

Upgrade from 0.6 to 0.7

On upgrade to version 0.7 you should run `./manage.py syncdb` to setup new tables and `./manage.py rebuild_index` to build index for fulltext search.

Upgrade from 0.7 to 0.8

On upgrade to version 0.8 you should run `./manage.py syncdb` to setup new tables, `./manage.py setupgroups` to update privileges setup and `./manage.py rebuild_index` to rebuild index for fulltext search.

Upgrade from 0.8 to 0.9

On upgrade to version 0.9 file structure has changed. You need to move `repos` and `whoosh-index` to `weblate` folder. Also running `./manage.py syncdb`, `./manage.py setupgroups` and `./manage.py setuplang` is recommended to get latest updates of privileges and language definitions.

Upgrade from 0.9 to 1.0

On upgrade to version 1.0 one field has been added to database, you need to invoke following SQL command to adjust it:

```
ALTER TABLE `trans_subproject` ADD `template` VARCHAR(200);
```

Upgrade from 1.0 (1.1) to 1.2

On upgrade to version 1.2, the migration procedure has changed. It now uses South for migrating database. To switch to this new migration schema, you need to run following commands:

```
./manage.py syncdb
./manage.py migrate trans 0001 --fake
./manage.py migrate accounts 0001 --fake
./manage.py migrate lang 0001 --fake
```

Also please note that there are several new requirements and version 0.8 of django-registration is now being required, see *Requirements* for more details.

Once you have done this, you can use *Generic upgrade instructions*.

Upgrade from 1.2 to 1.3

Since 1.3, `settings.py` is not shipped with Weblate, but only example settings as `settings_example.py` it is recommended to use it as new base for your setup.

Upgrade from 1.4 to 1.5

Several internal modules and paths have been renamed and changed, please adjust your `settings.py` to match that (consult `settings_example.py` for correct values).

- Many modules lost their `weblate.` prefix.
- Checks were moved to submodules.
- Locales were moved to top level directory.

The migration of database structure to 1.5 might take quite long, it is recommended to put your site offline, while the migration is going on.

Note: If you have update in same directory, stale `*.pyc` files might be left around and cause various import errors. To recover from this, delete all of them in Weblate's directory, for example by `find . -name '*.pyc' -delete`.

Upgrade from 1.6 to 1.7

The migration of database structure to 1.7 might take quite long, it is recommended to put your site offline, while the migration is going on.

If you are translating monolingual files, it is recommended to rerun quality checks as they might have been wrongly linked to units in previous versions.

Upgrade from 1.7 to 1.8

The migration of database structure to 1.8 might take quite long, it is recommended to put your site offline, while the migration is going on.

Authentication setup has been changed and some internal modules have changed name, please adjust your `settings.py` to match that (consult `settings_example.py` for correct values).

Also please note that there are several new requirements, see *Requirements* for more details.

4.3.2 Migrating from Pootle

As Weblate was originally written as replacement from Pootle, it is supported to migrate user accounts from Pootle. All you need to do is to copy `auth_user` table from Pootle, user profiles will be automatically created for users as they log in and they will be asked to update their settings. Alternatively you can use `importusers` to import dumped user credentials.

4.4 Authentication

4.4.1 User registration

The default setup for Weblate is to use python-social-auth for handling new users. This allows them to register using form on the website and after confirming their email they can contribute or by using some third party service to authenticate.

You can also completely disable new users registration using `REGISTRATION_OPEN`.

4.4.2 Authentication backends

By default Weblate uses Django built-in authentication and includes various social authentication options. Thanks to using Django authentication, you can also import user database from other Django based projects (see *Migrating from Pootle*).

Django can be additionally configured to authenticate against other means as well.

Social authentication

Thanks to python-social-auth, Weblate support authentication using many third party services such as Facebook, GitHub, Google or Bitbucket.

Please check their documentation for generic configuration instructions:

http://psa.matiasaguirre.net/docs/configuration/django.html

Note: By default, Weblate relies on third-party authentication services to provide validated email address, in case some of services you want to use do not support this, please remove `social.pipeline.social_auth.associate_by_email` from `SOCIAL_AUTH_PIPELINE` settings.

Enabling individual backends is quite easy, it's just matter of adding entry to `AUTHENTICATION_BACKENDS` setting and possibly adding keys needed for given authentication. Please note that some backends do not provide user email by default, you have to request it explicitely, otherwise Weblate will not be able to properly credit users contributions.

For example, enabling authentication against GitHub:

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social.backends.github.GithubOAuth2',
    'social.backends.email.EmailAuth',
    'accounts.auth.WeblateUserBackend',
)

# Social auth backends setup
SOCIAL_AUTH_GITHUB_KEY = 'GitHub Client ID'
```

```
SOCIAL_AUTH_GITHUB_SECRET = 'GitHub Client Secret'
SOCIAL_AUTH_GITHUB_SCOPE = ['user:email']
```

See Also:

http://psa.matiasaguirre.net/docs/backends/index.html

LDAP authentication

LDAP authentication can be best achieved using *django-auth-ldap* package. You can install it by usual means:

```
# Using PyPI
pip install django-auth-ldap

# Using apt-get
apt-get install python-django-auth-ldap
```

Once you have the package installed, you can hook it to Django authentication:

```
# Add LDAP backed, keep Django one if you want to be able to login
# even without LDAP for admin account
AUTHENTICATION_BACKENDS = (
    'django_auth_ldap.backend.LDAPBackend',
    'django.contrib.auth.backends.ModelBackend',
)

# LDAP server address
AUTH_LDAP_SERVER_URI = 'ldaps://ldap.example.net'

# DN to use for authentication
AUTH_LDAP_USER_DN_TEMPLATE = 'cn=%(user)s,o=Example'
# Depending on your LDAP server, you might use different DN
# like:
# AUTH_LDAP_USER_DN_TEMPLATE = 'ou=users,dc=example,dc=com'

# List of attributes to import from LDAP on login
AUTH_LDAP_USER_ATTR_MAP = {
    'first_name': 'givenName',
    'last_name': 'sn',
    'email': 'mail',
}
```

See Also:

http://pythonhosted.org/django-auth-ldap/

4.4.3 Access control

Weblate uses privileges system based on Django. The default setup (after you run `setupgroups`) consists of three groups *Guests*, *Users* and *Managers* which have privileges as described above. All new users are automatically added to *Users* group. The *Guests* groups is used for not logged in users.

Basically *Users* are meant as regular translators and *Managers* for developers who need more control over the translation - they can force committing changes to git, push changes upstream (if Weblate is configured to do so) or disable translation (eg. when there are some major changes happening upstream).

To customize this setup, it is recommended to remove privileges from *Users* group and create additional groups with finer privileges (eg. *Translators* group, which will be allowed to save translations and manage suggestions) and add selected users to this group. You can do all this from Django admin interface.

To completely lock down your Weblate installation you can use `LOGIN_REQUIRED_URLS` for forcing users to login and `REGISTRATION_OPEN` for disallowing new registrations.

Extra privileges

Weblate defines following extra privileges:

Can upload translation [Users, Managers] Uploading of translation files.

Can overwrite with translation upload [Users, Managers] Overwriting existing translations by uploading translation file.

Can define author of translation upload [Managers] Allows to define custom authorship when uploading translation file.

Can force committing of translation [Managers] Can force Git commit in the web interface.

Can see git repository URL [Users, Managers, Guests] Can see Git repository URL inside Weblate

Can update translation from git [Managers] Can force Git pull in the web interface.

Can push translations to remote git [Managers] Can force Git push in the web interface.

Can do automatic translation using other project strings [Managers] Can do automatic translation based on strings from other subprojects.

Can lock whole translation project [Managers] Can lock translation for updates, useful while doing some major changes in the project.

Can reset translations to match remote git [Managers] Can reset Git repository to match remote git.

Can save translation [Users, Managers] Can save translation (might be disabled with *Suggestion voting*).

Can accept suggestion [Users, Managers] Can accept suggestion (might be disabled with *Suggestion voting*).

Can delete suggestion [Users, Managers] Can delete suggestion (might be disabled with *Suggestion voting*).

Can vote for suggestion [Users, Managers] Can vote for suggestion (see *Suggestion voting*).

Can override suggestion state [Managers] Can save translation, accept or delete suggestion when automatic accepting by voting for suggestions is enabled (see *Suggestion voting*).

Can import dictionary [Users, Managers] Can import dictionary from translation file.

Can add dictionary [Users, Managers] Can add dictionary entries.

Can change dictionary [Users, Managers] Can change dictionary entries.

Can delete dictionary [Users, Managers] Can delete dictionary entries.

Can lock translation for translating [Users, Managers] Can lock translation while translating (see *Translation locking*).

Can add suggestion [Users, Managers, Guests] Can add new suggestions.

Can use machine translation [Users, Managers] Can use machine translations (see *Machine translation setup*).

Per project access control

New in version 1.4: This feature is available since Weblate 1.4.

Note: By enabling ACL, all users are prohibited to access anything within given project unless you add them the permission to do that.

Additionally you can limit users access to individual projects. This feature is enabled by *Enable ACL* at Project configuration. Once you enable this, users without specific privilege (*trans | project | Can access project NAME*) can not access this project.

To allow access to this project, you have to add the privilege to do so either directly to given user or group of users in Django admin interface.

See Also:

https://docs.djangoproject.com/en/1.4/topics/auth/default/#auth-admin

4.5 Translation projects

4.5.1 Translation organization

Weblate organizes translatable content into tree like structure. The toplevel object is *Project*, which should hold all translations which belong together (for example translation of an application in several versions and/or documentation). On the next level, there is *Subproject*, which is actually the resource to translate. Here you define Git repository to use and mask of files to translate. Bellow *Subproject* there are individual translations, which are handled automatically by Weblate as the translation files (matching mask defined in *Subproject*) appear in Git repository.

4.5.2 Administration

Administration of Weblate is done through standard Django admin interface, which is available under `/admin/` URL.

4.5.3 Adding new resources

All translation resources need to be available as Git repositories and are organized as project/subproject structure.

Weblate supports wide range of translation formats supported by translate toolkit, see *Supported formats* for more information.

Monolingual resources

Weblate does support both multilingual and monolingual formats. For easier translating of monolingual formats, you should provide template file, which contains mapping of message IDs to source language (usually English).

4.5.4 Project

To add new resource to translate, you need to create translation project first. The project is sort of shelf, in which real translations are folded. All subprojects in same project share suggestions and dictionary, also the translations are automatically propagated through the all subproject in single project (unless disabled in subproject configuration).

The project has only few attributes giving translators information about project.

Commit message

The commit message on each commit Weblate does, it can use following format strings in the message:

%(language)s Language code

%(language_name)s Language name

%(subproject)s Subproject name

%(project)s Project name

%(total)s Total strings count

%(fuzzy)s Fuzzy strings count

%(fuzzy_percent)s Fuzzy strings percent

%(translated)s Translated strings count

%(translated_percent)s Translated strings percent

Adjusting interaction

There are also additional features which you can control, like automatic pushing of changes (see also *Pushing changes*), merge or rebase (see *Merge or rebase*), git committer name or maintaining of Translation-Team header.

4.5.5 Subproject

Subproject is real resource for translating. You enter Git repository location and file mask which files to translate and Weblate automatically fetches the Git and finds all matching translatable files.

Should the language definition for translation be missing, empty definition is created and named as "cs_CZ (generated)". You should adjust the definition and report this back to Weblate authors so that missing language can be included in next release.

The subproject contains all important parameters for working with Git and getting translations out of it:

Repo Git repository used to pull changes.

This can be either real Git URL or `weblate://project/subproject` indicating that Git repository should be shared with another subproject.

Push Git URL used for pushing, this is completely optional and push support will be disabled when this is empty.

Repoweb URL of repository browser to display source files (location where messages are used). When empty no such links will be generated.

For example on GitHub, you would use something like `https://github.com/nijel/weblate-hello/blob/%(bran`

Branch Which branch to checkout from the Git and where to look for translations.

Filemask Mask of files to translate including path. It should include one * replacing language code. In case your Git repository contains more than one translation files (eg. more Gettext domains), you need to create separate subproject for each. For example `po/*.po` or `locale/*/LC_MESSAGES/django.po`.

Monolingual base language file Base file containing strings definition for *Monolingual resources*.

Base file for new translations Base file used to generate new translations, eg. `.pot` file with Gettext.

Report source bugs Email address used for reporting upstream bugs. This address will also receive notification about any source string comments made in Weblate.

Locked You can lock the translation to prevent updates by users.

Allow translation propagation You can disable propagation of translations to this subproject from other subprojects within same project. This really depends on what you are translating, sometimes it's desirable to have same string used.

Pre commit script One of scripts defined in `PRE_COMMIT_SCRIPTS` which is executed before commit.

Extra commit file Additional file to include in commit, usually this one is generated by pre commit script described above.

Save translation history Whether to store history of translation changes in database.

Suggestion voting Enable voting for suggestions, see *Suggestion voting*.

Autoaccept suggestions Automatically accept voted suggestions, see *Suggestion voting*.

Quality checks flags Additional flags to pass to quality checks, see *Customizing checks*.

See Also:

Does Weblate support other VCS than Git?, *Pre commit processing of translations*

4.5.6 Importing speed

Fetching Git repository and importing translations to Weblate can be lengthy process depending on size of your translations. Here are some tips to improve this situation:

Clone Git repository in advance

You can put in place Git repository which will be used by Weblate. The repositories are stored in path defined by `GIT_ROOT` in `settings.py` in `<project>/<subproject>` directories.

This can be especially useful if you already have local clone of this repository and you can use `--reference` option while cloning:

```
git clone \
    --reference /path/to/checkout \
    git://github.com/nijel/weblate.git \
    weblate/repos/project/subproject
```

Optimize configuration

The default configuration is useful for testing and debugging Weblate, while for production setup, you should do some adjustments. Many of them have quite big impact on performance. Please check *Production setup* for more details, especially:

- *Enable indexing offloading*
- *Enable caching*
- *Use powerful database engine*
- *Disable debug mode*

Disable not needed checks

Some quality checks can be quite expensive and if you don't need them, they can save you some time during import. See `CHECK_LIST` for more information how to configure this.

4.5.7 Automatic creation of subprojects

In case you have project with dozen of po files, you might want to import all at once. This can be achieved using `import_project`.

First you need to create project which will contain all subprojects and then it's just a matter of running `import_project`.

See Also:

Management commands

4.5.8 Accessing repositories

Private repositories

In case you want Weblate to access private repository it needs to get to it somehow. Most frequently used method here is based on SSH. To have access to such repository, you generate SSH key for Weblate and authorize it to access the repository.

You also need to verify SSH host keys of servers you are going to access.

You can generate or display key currently used by Weblate in the admin interface (follow *SSH keys* link on main admin page).

Note: The keys need to be without password to make it work, so be sure they are well protected against malicious usage.

Using proxy

If you need to access http/https Git repositories using a proxy server, you need to configure Git to use it.

This can be configured using the `http_proxy`, `https_proxy`, and `all_proxy` environment variables (check cURL documentation for more details) or by enforcing it in Git configuration, for example:

```
git config --global http.proxy http://user:password@proxy.example.com:80
```

Note: The proxy setting needs to be done in context which is used to execute Weblate. For the environment it should be set for both server and cron jobs. The Git configuration has to be set for the user which is running Weblate.

See Also:

http://curl.haxx.se/docs/manpage.html, http://git-scm.com/docs/git-config

4.5.9 Fulltext search

Fulltext search is based on Whoosh. You can either allow Weblate to directly update index on every change to content or offload this to separate process by `OFFLOAD_INDEXING`.

The first approach (immediate updates) allows more up to date index, but suffers locking issues in some setup (eg. Apache's mod_wsgi) and produces more fragmented index.

Offloaded indexing is always better choice for production setup - it only marks which items need to be reindexed and you need to schedule background process (`update_index`) to update index. This leads to faster response of the site and less fragmented index with cost that it might be slightly outdated.

See Also:

`update_index`, `OFFLOAD_INDEXING`, *Fulltext search is too slow*, *I get "Lock Error" quite often while translating*, *Rebuilding index has failed with "No space left on device"*

4.6 Continuous translation

Weblate provides you great infrastructure for translation to closely follow your development. This way translators can work on translations whole time and are not forced to translate huge amount of new texts before release.

4.6.1 Updating repositories

You should set up some way how backend repositories are updated from their source. You can either use hooks (see *Notification hooks*) or just regularly run `updategit --all`.

With Gettext po files, you might be often bitten by conflict in PO file headers. To avoid it, you can use shipped merge driver (`examples/git-merge-gettext-po`). To use it just put following configuration to your `.gitconfig`:

```
[merge "merge-gettext-po"]
  name = merge driver for gettext po files
  driver = /path/to/weblate/examples/git-merge-gettext-po %O %A %B
```

And enable it's use by defining proper attributes in given repository (eg. in `.git/info/attribute`):

```
*.po merge=merge-gettext-po
```

Note: This merge driver assumes the changes in POT files always are done in branch we're trying to merge.

See Also:

http://www.no-ack.org/2010/12/writing-git-merge-driver-for-po-files.html

4.6.2 Pushing changes

Each project can have configured push URL and in such case Weblate offers button to push changes to remote repository in web interface.

I case you will use SSH for pushing, you need to have key without passphrase (or use ssh-agent for Django) and the remote server needs to be verified by you first, otherwise push will fail.

Note: You can also enable automatic pushing changes on commit, this can be done in project configuration.

See Also:

Private repositories for setting up SSH keys

4.6.3 Merge or rebase

By default Weblate merges upstream repository into it's own. This is safest way in case you also access underlying repository by other means. In case you don't need this, you can enable rebasing of changes on upstream, what will produce history with less merge commits.

Note: Rebasing can cause you troubles in case of complicated merges, so carefully consider whether you want to enable them or not.

4.6.4 Interacting with others

Weblate makes it easy to interact with others using it's API.

See Also:

Weblate's Web API

4.6.5 Lazy commits

Default behaviour (configured by `LAZY_COMMITS`) of Weblate is to group commits from same author into one if possible. This heavily reduces number of commits, however you might need to explicitly tell to do the commits in case you want to get Git repository in sync, eg. for merge (this is by default allowed for Managers group, see *Access control*).

The changes are in this mode committed once any of following conditions is fulfilled:

- somebody else works on the translation
- merge from upstream occurs
- import of translation happens
- translation for a language is completed
- explicit commit is requested

You can also additionally set a cron job to commit pending changes after some delay, see `commit_pending`.

4.6.6 Pre commit processing of translations

In many cases you might want to automatically do some changes to translation before it is committed to the repository. The pre commit script is exactly the place to achieve this.

Before using any scripts, you need to list them in `PRE_COMMIT_SCRIPTS` configuration variable. Then you can enable them at *Subproject* configuration as *Pre commit script*.

The hook script is executed using system() call, so it is evaluated in a shell. It is passed single parameter consisting of file name of current translation.

The script can also generate additional file to be included in the commit. This can be configured as *Extra commit file* at *Subproject* configuration. You can use following format strings in the filename:

`%(language)s` Language code

Example - generating mo files in repository

Allow usage of the hook in the configuration

```
PRE_COMMIT_SCRIPTS = (
    '/usr/share/weblate/examples/hook-generate-mo',
)
```

To enable it, choose now *hook-generate-mo* as *Pre commit script*. You will also want to add path to generated files to be included in Git commit, for example `po/%(language)s.mo` as *Extra commit file*.

You can find more example scripts in `examples` folder within Weblate sources, their name start with `hook-`.

4.7 Translation process

4.7.1 Suggestion voting

New in version 1.6: This feature is available since Weblate 1.6. In default Weblate setup, everybody can add suggestions and logged in users can accept them. You might however want to have more eyes on the translation and require more people to accept them. This can be achieved by suggestion voting. You can enable this on *Subproject* configuration by *Suggestion voting* and *Autoaccept suggestions*. The first one enables voting feature, while the latter allows you to configure threshold at which suggestion will gets automatically accepted (this includes own vote from suggesting user).

Note: Once you enable automatic accepting, normal users lose privilege to directly save translations or accept suggestions. This can be overriden by *Can override suggestion state* privilege (see *Access control*).

You can combine these with *Access control* into one of following setups:

- Users can suggest and vote for suggestions, limited group controls what is accepted - enable voting but not automatic accepting and remove privilege from users to save translations.
- Users can suggest and vote for suggestions, which get automatically accepted once defined number of users agree on this - enable voting and set desired number of votes for automatic accepting.
- Optional voting for suggestions - you can also only enable voting and in this case it can be optionally used by users when they are not sure about translation (they can suggest more of them).

4.7.2 Translation locking

To improve collaboration, it is good to prevent duplicate effort on translation. To achieve this, translation can be locked for single translator. This can be either done manually on translation page or is done automatically when somebody starts to work on translation. The automatic locking needs to be enabled using `AUTO_LOCK`.

The automatic lock is valid for `AUTO_LOCK_TIME` seconds and is automatically extended on every translation made and while user has opened translation page.

User can also explicitly lock translation for `LOCK_TIME` seconds.

4.8 Checks and fixups

4.8.1 Custom automatic fixups

You can also implement own automatic fixup in addition to standard ones and include them in `AUTOFIX_LIST`.

The automatic fixes are powerful, but can also cause damage, be careful when writing one.

For example following automatic fixup would replace every occurrence of string `foo` in translation with `bar`:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2013 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <http://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
#

from trans.autofixes.base import AutoFix
from django.utils.translation import ugettext_lazy as _


class ReplaceFooWithBar(AutoFix):
    '''
    Replaces foo with bar.
    '''

    name = _('Foobar')

    def fix_single_target(self, target, source, unit):
        if 'foo' in target:
            return target.replace('foo', 'bar'), True
        return target, False
```

4.8.2 Customizing checks

Fine tuning existing checks

In *Subproject* setup, you can fine tune some of the checks, here is current list of flags accepted:

`rst-text` Treat text as RST document, affects *Not translated*.

`python-format`, `c-format`, `php-format`, `python-brace-format` Treats all string like format strings, affects *Python format*, *C format*, *PHP format*, *Python brace format*, *Not translated*.

ignore-* Ignores given check for a subproject.

These flags are understood both in *Subproject* settings and in translation file itself (eg. in GNU Gettext).

Writing own checks

Weblate comes with wide range of quality checks (see *Quality checks*), though they might not 100% cover all you want to check. The list of performed checks can be adjusted using `CHECK_LIST` and you can also add custom checks. All you need to do is to subclass `trans.checks.Check`, set few attributes and implement either `check` or `check_single` methods (first one if you want to deal with plurals in your code, the latter one does this for you). You will find below some examples.

Checking translation text does not contain "foo"

This is pretty simple check which just checks whether translation does not contain string "foo".

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2013 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <http://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
'''
Simple quality check example.
'''

from trans.checks import TargetCheck
from django.utils.translation import ugettext_lazy as _


class FooCheck(TargetCheck):

    # Used as identifier for check, should be unique
    check_id = 'foo'

    # Short name used to display failing check
    name = _('Foo check')

    # Description for failing check
    description = _('Your translation is foo')

    # Real check code
```

```
    def check_single(self, source, target, unit):
        return 'foo' in target
```

Checking Czech translation text plurals differ

Check using language information to verify that two plural forms in Czech language are not same.

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2013 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <http://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
'''
Quality check example for Czech plurals.
'''

from trans.checks import TargetCheck
from django.utils.translation import ugettext_lazy as _


class PluralCzechCheck(TargetCheck):

    # Used as identifier for check, should be unique
    check_id = 'foo'

    # Short name used to display failing check
    name = _('Foo check')

    # Description for failing check
    description = _('Your translation is foo')

    # Real check code
    def check(self, sources, targets, unit):
        if self.is_language(unit, ['cs']):
            return targets[1] == targets[2]
        return False
```

4.9 Machine translation

4.9.1 Machine translation setup

Weblate has builtin support for several machine translation services and it's up to administrator to enable them. The services have different terms of use, so please check whether you are allowed to use them before enabling in Weblate. The individual services are enabled using `MACHINE_TRANSLATION_SERVICES`.

The source langauge can be configured by `SOURCE_LANGUAGE` and is shared for all translations within Weblate.

Amagama

Special installation of *tmserver* run by Virtaal authors.

To enable this service, add `trans.machine.tmserver.AmagamaTranslation` to `MACHINE_TRANSLATION_SERVICES`.

See Also:

http://docs.translatehouse.org/projects/virtaal/en/latest/amagama.html

Apertium

A free/open-source machine translation platform providing translation to limited set of languages.

You should get API key from them, otherwise number of requests is rate limited.

To enable this service, add `trans.machine.apertium.ApertiumTranslation` to `MACHINE_TRANSLATION_SERVICES`.

See Also:

`MT_APERTIUM_KEY`, http://www.apertium.org/

Glosbe

Free dictionary and translation memory for almost every living language.

API is free to use, regarding indicated data source license. There is a limit of call that may be done from one IP in fixed period of time, to prevent from abuse.

To enable this service, add `trans.machine.glosbe.GlosbeTranslation` to `MACHINE_TRANSLATION_SERVICES`.

See Also:

http://glosbe.com/

Google Translate

Machine translation service provided by Google.

This service uses Translation API and you need to obtain API key and enable billing on Google API console.

To enable this service, add `trans.machine.google.GoogleTranslation` to `MACHINE_TRANSLATION_SERVICES`.

See Also:

MT_GOOGLE_KEY, https://developers.google.com/translate/

Google Web Translate

Machine translation service provided by Google.

Please note that this does not use official Translation API but rather web based translation interface.

To enable this service, add `trans.machine.google.GoogleWebTranslation` to MACHINE_TRANSLATION_SERVICES.

See Also:

http://translate.google.com/

Microsoft Translator

Machine translation service provided by Microsoft.

You need to register at Azure market and use Client ID and secret from there.

To enable this service, add `trans.machine.microsoft.MicrosoftTranslation` to MACHINE_TRANSLATION_SERVICES.

See Also:

MT_MICROSOFT_ID, MT_MICROSOFT_SECRET, http://www.microsofttranslator.com/, https://datamarket.azure.com/developer/applications/

MyMemory

Huge translation memory with machine translation.

Free, anonymous usage is currently limited to 100 requests/day, or to 1000 requests/day when you provide contact email in MT_MYMEMORY_EMAIL. you can also ask them for more.

To enable this service, add `trans.machine.mymemory.MyMemoryTranslation` to MACHINE_TRANSLATION_SERVICES.

See Also:

MT_MYMEMORY_EMAIL, MT_MYMEMORY_USER, MT_MYMEMORY_KEY, http://mymemory.translated.net/

Open-Tran

Database of open source translations.

To enable this service, add `trans.machine.opentran.OpenTranTranslation` to MACHINE_TRANSLATION_SERVICES.

See Also:

http://www.open-tran.eu/

tmserver

You can run your own translation memory server which is bundled with Translate-toolkit and let Weblate talk to it. You can also use it with amaGama server, which is enhanced version of tmserver.

First you will want to import some data to the translation memory:

To enable this service, add `trans.machine.tmserver.TMServerTranslation` to `MACHINE_TRANSLATION_SERVICES`.

```
build_tmdb -d /var/lib/tm/db -s en -t cs locale/cs/LC_MESSAGES/django.po
build_tmdb -d /var/lib/tm/db -s en -t de locale/de/LC_MESSAGES/django.po
build_tmdb -d /var/lib/tm/db -s en -t fr locale/fr/LC_MESSAGES/django.po
```

Now you can start tmserver to listen to your requests:

```
tmserver -d /var/lib/tm/db
```

And configure Weblate to talk to it:

```
MT_TMSERVER = 'http://localhost:8888/'
```

See Also:

`MT_TMSERVER`, http://docs.translatehouse.org/projects/translate-toolkit/en/latest/commands/tmserver.html, http://amagama.translatehouse.org/

Weblate

Weblate can be source of machine translation as well. There are two services to provide you results - one does exact search for string, the other one finds all similar strings.

First one is useful for full string translations, the second one for finding individual phrases or words to keep the translation consistent.

To enable these services, add `trans.machine.weblatetm.WeblateSimilarTranslation` (for similar string matching) and/or `trans.machine.weblatetm.WeblateTranslation` (for exact string matching) to `MACHINE_TRANSLATION_SERVICES`.

Note: For similarity matching, it is recommended to have Whoosh 2.5.2 or later, earlier versions can cause infinite looks under some occasions.

4.9.2 Custom machine translation

You can also implement own machine translation services using few lines of Python code. Following example implements translation to fixed list of languages using `dictionary` Python module:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2013 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <http://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
```

```
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
'''
Machine translation example.
'''

from trans.machine.base import MachineTranslation
import dictionary


class SampleTranslation(MachineTranslation):
    '''
    Sample machine translation interface.
    '''
    name = 'Sample'

    def download_languages(self):
        '''
        Returns list of languages your machine translation supports.
        '''
        return set(('cs',))

    def download_translations(self, language, text, unit, request):
        '''
        Returns tuple with translations.
        '''
        return [(t, 100, self.name, text) for t in dictionary.translate(text)]
```

You can list own class in `MACHINE_TRANSLATION_SERVICES` and Weblate will start using that.

4.10 Configuration

All settings are stored in `settings.py` (as usual for Django).

Note: After changing any of these settings, you need to restart Weblate. In case it is run as mod_wsgi, you need to restart Apache to reload the configuration.

See Also:

Please check also Django's documentation for parameters which configure Django itself.

4.10.1 ANONYMOUS_USER_NAME

User name of user for defininining privileges of not logged in user.

See Also:

Access control

4.10.2 AUTO_LOCK

Enables automatic locking of translation when somebody is working on it.

See Also:

Translation locking

4.10.3 AUTO_LOCK_TIME

Time in seconds for how long the automatic lock for translation will be active.

See Also:

Translation locking

4.10.4 AUTOFIX_LIST

List of automatic fixups to apply when saving the message.

Available fixes:

trans.autofixes.whitespace.SameBookendingWhitespace Fixes up whitespace in beginning and end of the string to match source.

trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis Replaces traling dots with ellipsis if source string has it.

trans.autofixes.chars.RemoveZeroSpace Removes zero width space char if source does not contain it.

For example you can enable only few of them:

```
AUTOFIX_LIST = (
    'trans.autofixes.whitespace.SameBookendingWhitespace',
    'trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis',
)
```

See Also:

Automatic fixups, *Custom automatic fixups*

4.10.5 BACKGROUND_HOOKS

Whether to run hooks in background. This is generally recommended unless you are debugging.

4.10.6 CHECK_LIST

List of quality checks to perform on translation.

Some of the checks are not useful for all projects, so you are welcome to adjust list of performed on your installation.

For example you can enable only few of them:

```
CHECK_LIST = (
    'trans.checks.same.SameCheck',
    'trans.checks.format.CFormatCheck',
    'trans.checks.chars.ZeroWidthSpaceCheck',
)
```

See Also:

Quality checks, *Customizing checks*

4.10.7 ENABLE_HOOKS

Whether to enable anonymous remote hooks.

See Also:

Notification hooks

4.10.8 GIT_ROOT

Path where Weblate will store cloned Git repositories. Defaults to `repos` subdirectory.

4.10.9 LAZY_COMMITS

Delay creating Git commits until this is necessary. This heavily reduces number of commits generated by Weblate at expense of temporarily not being able to merge some changes as they are not yet committed.

See Also:

Lazy commits

4.10.10 LOCK_TIME

Time in seconds for how long the translation will be locked for single translator when locked manually.

See Also:

Translation locking

4.10.11 LOGIN_REQUIRED_URLS

List of URL which require login (besides standard rules built into Weblate). This allows you to password protect whole installation using:

```
LOGIN_REQUIRED_URLS = (
    r'/(.*)$',
)
```

4.10.12 LOGIN_REQUIRED_URLS_EXCEPTIONS

List of exceptions for `LOGIN_REQUIRED_URLS`, in case you won't specify this list, the default value will be used, which allows users to access login page.

Some of exceptions you might want to include:

```
LOGIN_REQUIRED_URLS_EXCEPTIONS = (
    r'/accounts/(.*)$', # Required for login
    r'/media/(.*)$',    # Required for development mode
    r'/widgets/(.*)$',  # Allowing public access to widgets
    r'/data/(.*)$',     # Allowing public access to data exports
```

```
    r'/hooks/(.*)$',    # Allowing public access to notification hooks
)
```

4.10.13 MACHINE_TRANSLATION_SERVICES

List of enabled machine translation services to use.

Note: Many of services need additional configuration like API keys, please check their documentation for more details.

```
MACHINE_TRANSLATION_SERVICES = (
    'trans.machine.apertium.ApertiumTranslation',
    'trans.machine.glosbe.GlosbeTranslation',
    'trans.machine.google.GoogleTranslation',
    'trans.machine.microsoft.MicrosoftTranslation',
    'trans.machine.mymemory.MyMemoryTranslation',
    'trans.machine.opentran.OpenTranTranslation',
    'trans.machine.tmserver.TMServerTranslation',
    'trans.machine.weblatetm.WeblateSimilarTranslation',
    'trans.machine.weblatetm.WeblateTranslation',
)
```

See Also:

Machine translation setup, *Machine translation*

4.10.14 MT_APERTIUM_KEY

API key for Apertium Web Service, you can register at http://api.apertium.org/register.jsp

See Also:

Apertium, *Machine translation setup*, *Machine translation*

4.10.15 MT_GOOGLE_KEY

API key for Google Translate API, you can register at https://developers.google.com/translate/

See Also:

Google Translate, *Machine translation setup*, *Machine translation*

4.10.16 MT_MICROSOFT_ID

Cliend ID for Microsoft Translator service.

See Also:

Microsoft Translator, *Machine translation setup*, *Machine translation*, https://datamarket.azure.com/developer/applications/

4.10.17 MT_MICROSOFT_SECRET

Client secret for Microsoft Translator service.

See Also:

Microsoft Translator, *Machine translation setup*, *Machine translation*, https://datamarket.azure.com/developer/applications/

4.10.18 MT_MYMEMORY_EMAIL

MyMemory identification email, you can get 1000 requests per day with this.

See Also:

MyMemory, *Machine translation setup*, *Machine translation*, http://mymemory.translated.net/doc/spec.php

4.10.19 MT_MYMEMORY_KEY

MyMemory access key for private translation memory, use together with `MT_MYMEMORY_USER`.

See Also:

MyMemory, *Machine translation setup*, *Machine translation*, http://mymemory.translated.net/doc/keygen.php

4.10.20 MT_MYMEMORY_USER

MyMemory user id for private translation memory, use together with `MT_MYMEMORY_KEY`.

See Also:

MyMemory, *Machine translation setup*, *Machine translation*, http://mymemory.translated.net/doc/keygen.php

4.10.21 MT_TMSERVER

URL where tmserver is running.

See Also:

tmserver, *Machine translation setup*, *Machine translation*, http://docs.translatehouse.org/projects/translate-toolkit/en/latest/commands/tmserver.html

4.10.22 NEARBY_MESSAGES

How many messages around current one to show during translating.

4.10.23 OFFLOAD_INDEXING

Offload updating of fulltext index to separate process. This heavily improves responsiveness of online operation on expense of slightly outdated index, which might still point to older content.

While enabling this, don't forget scheduling runs of `update_index` in cron or similar tool.

This is recommended setup for production use.

See Also:

Fulltext search

4.10.24 PRE_COMMIT_SCRIPTS

List of scripts which are allowed as pre commit scripts. The script needs to be later enabled in subproject configuration.

For example you can allow script which does some cleanup:

```
PRE_COMMIT_SCRIPTS = (
    '/usr/local/bin/cleanup-translation',
)
```

Note: The hook is executed using system() call, so it is evaluated in a shell.

See Also:

Pre commit processing of translations

4.10.25 REGISTRATION_CAPTCHA

A boolean (either `True` or `False`) indicating whether registration of new accounts is protected by captcha. This setting is optional, and a default of True will be assumed if it is not supplied.

4.10.26 REGISTRATION_OPEN

A boolean (either `True` or `False`) indicating whether registration of new accounts is currently permitted. This setting is optional, and a default of True will be assumed if it is not supplied.

4.10.27 SITE_TITLE

Site title to be used in website and emails as well.

4.10.28 SOURCE_LANGUAGE

Source language used for translation. This is mostly useful for machine translation services.

4.10.29 TTF_PATH

Path to Droid fonts used for widgets and charts.

4.10.30 WHOOSH_INDEX

Directory where Whoosh fulltext indices will be stored. Defaults to `whoosh-index` subdirectory.

4.11 Management commands

Note: Running management commands under different user than is running your webserver can cause wrong permissions on some files, please check *Filesystem permissions* for more details.

The ./manage.py is extended with following commands:

4.11.1 checkgit <project|project/subproject>

manage.py checkgit

Prints current state of backend git repository.

You can either define which project or subproject to update (eg. `weblate/master`) or use `--all` to update all existing subprojects.

4.11.2 commitgit <project|project/subproject>

manage.py commitgit

Commits any possible pending changes to backend git repository.

You can either define which project or subproject to update (eg. `weblate/master`) or use `--all` to update all existing subprojects.

4.11.3 commit_pending <project|project/subproject>

manage.py commit_pending

Commits pending changes older than given age (using `--age` parameter, defaults to 24 hours).

You can either define which project or subproject to update (eg. `weblate/master`) or use `--all` to update all existing subprojects.

This is most useful if executed periodically from cron or similar tool:

```
./manage.py commit_pending --all --age=48
```

4.11.4 cleanuptrans

manage.py cleanuptrans

Cleanups orphaned checks and translation suggestions.

4.11.5 createadmin

manage.py createadmin

Creates `admin` account with password `admin`.

4.11.6 import_project <project> <gitrepo> <branch> <filemask>

manage.py import_project

Batch imports subprojects into project based on file mask.

<project> names an existing project, into which the subprojects should be imported.

The *<gitrepo>* defines URL of Git repository to use, and *<branch>* the git branch. To import additional translation subprojects, from an existing Weblate subproject, use a *weblate://<project>/<subproject>* URL for the *<gitrepo>*.

The repository is searched for directories matching a double wildcard (**) in the *<filemask>*. Each of these is then added as a subproject, named after the matched directory. Existing subprojects will be skipped.

To customise the subproject's name, use the `--name-template` option. Its parameter is a python formatting string, which will expect the match from *<filemask>*.

By format string passed by the `--base-file-template` option you can customize base file for monolingual translations.

You can also specify file format to use (see *Supported formats*) by the `--file-format` parameter. The default is autodetection.

For example:

```
./manage.py import_project debian-handbook git://anonscm.debian.org/debian-handbook/debian-handbook.g
```

4.11.7 importusers <file.json>

manage.py importusers

Imports users from JSON dump of Django auth_users database.

You can dump users from existing Django installation using:

```
./manage.py dumpdata auth.User > users.json
```

4.11.8 list_ignored_checks

manage.py list_ignored_checks

Lists most frequently ignored checks. This can be useful for tuning your setup, if users have to ignore too many of consistency checks.

4.11.9 list_versions

manage.py list_versions

Lists versions of Weblate dependencies.

4.11.10 loadpo <project|project/subproject>

manage.py loadpo

Reloads translations from disk (eg. in case you did some updates in Git repository).

You can use `--force` to force update even if the files should be up to date. Additionally you can limit languages to process with `--lang`.

You can either define which project or subproject to update (eg. `weblate/master`) or use `--all` to update all existing subprojects.

4.11.11 pushgit <project|project/subproject>

`manage.py pushgit`

Pushes committed changes to upstream Git repository. With `--force-commit` it also commits any pending changes.

You can either define which project or subproject to update (eg. `weblate/master`) or use `--all` to update all existing subprojects.

4.11.12 rebuild_index <project|project/subproject>

`manage.py rebuild_index`

Rebuilds index for fulltext search. This might be lengthy operation if you have huge set of translation units.

You can use `--clean` to remove all words from database prior updating.

See Also:

Fulltext search

4.11.13 update_index

`manage.py update_index`

Updates index for fulltext search when `OFFLOAD_INDEXING` is enabled.

It is recommended to run this frequently (eg. every 5 minutes) to have index uptodate.

See Also:

Fulltext search

4.11.14 setupgroups

`manage.py setupgroups`

Configures default groups and (if called with `--move`) assigns all users to default group.

The option `--no-update` disables update of existing groups (only adds new ones).

See Also:

Access control

4.11.15 setuplang

manage.py setuplang

Setups list of languages (it has own list and all defined in translate-toolkit).

The option `--no-update` disables update of existing languages (only adds new ones).

4.11.16 updatechecks <project|project/subproject>

manage.py updatechecks

Updates all check for all units. This could be useful only on upgrades which do major changes to checks.

You can either define which project or subproject to update (eg. `weblate/master`) or use `--all` to update all existing subprojects.

4.11.17 updategit <project|project/subproject>

manage.py updategit

Fetches remote Git repositories and updates internal cache.

You can either define which project or subproject to update (eg. `weblate/master`) or use `--all` to update all existing subprojects.

CHAPTER 5

Frequently Asked Questions

5.1 Configuration

5.1.1 How to create automatic worflow?

Weblate can handle all the translation things semi-automatically for you. If you will give it push access to your repository, the translations can live without interaction unless some merge conflict occurs.

1. Set up you git repository to tell Weblate whenever there is any change, see *Notification hooks* for information how to do it.
2. Set push URL at your *Subproject* in Weblate, this will allow Weblate to push changes to your repository.
3. Enable push on commit on your *Project* in Weblate, this will make Weblate push changes to your repository whenever they are committed at Weblate.
4. Optionally setup cron job for `commit_pending`.

See Also:

Continuous translation

5.1.2 How to access repositories over SSH?

Please see *Private repositories* for information about setting up SSH keys.

5.1.3 How to fix merge conflicts in translations?

The easiest way is to solve all conflicts locally at your workstation - simply add Weblate as remote repository, merge it into upstream and fix conflicts. Once you push changes back, Weblate will be able to use merged version without any other special actions.

```
# Add remote
git remote add weblate git://git.weblate.org/debian-handbook.git

# Update remotes
git remote update
```

```
# Merge Weblate changes
git merge weblate/master

# Resolve conflicts
edit ....
git add ...
...
git commit

# Push changes to upstream respository, Weblate will fetch merge from there
git push
```

See Also:

How to export Git repository weblate uses?

5.1.4 How do I translate several branches at once?

Weblate supports pushing translation changes within one *Project*. For every *Subproject* which has it enabled (the default behavior), the change made is automatically propagated to others. This way the translations are kept synchronized even if the branches themselves have already diverged quite a lot and it is not possible to simply merge translation changes between them.

Once you merge changes from Weblate, you might have to merge these branches (depending on your development workflow) discarding differences:

```
git merge -s ours origin/maintenance
```

5.1.5 How to export Git repository weblate uses?

There is nothing special about the repository, it lives under `GIT_ROOT` directory and is named as *project/subproject/*. If you have SSH access to this machine, you can use the repository directly.

For anonymous access you might want to run git server and let it serve the repository to outside world.

5.1.6 What are options of pushing changes back upstream?

This heavily depends on your setup, Weblate is quite flexible in this area. Here are examples of workflows used with Weblate:

- Weblate automatically pushes and merges changes (see *How to create automatic worflow?*)
- You tell manually Weblate to push (it needs push access to upstream repository)
- Somebody manually merges changes from Weblate git repository into upstream repository
- Somebody rewrites history produced by Weblate (eg. by eliminating merge commits), merges changes and tells Weblate to reset contet on upstream repository.

Of course your are free to mix all of these as you wish.

5.1.7 How can I check if my Weblate is configured properly?

Weblate includes set of configuration checks, which you can see in admin interface, just follow *Performace report* link in admin interface or directly open `/admin/performance/` URL.

5.1.8 Why does registration contain example.com as domain?

Weblate uses Django sites framework and it defines site name inside the database. Please open admin interface and edit default site name and domain (you can do that directly at `/admin/sites/site/1/` URL under your Weblate installation).

See Also:

https://docs.djangoproject.com/en/dev/ref/contrib/sites/

5.2 Usage

5.2.1 How do I review others translations?

- You can subscribe to any changes made in *Subscriptions* and then check other contributions in email.
- There is review tool available at bottom of translation view, where you can choose to browse translations made by others since given date.

5.2.2 How do I provide feedback on source string?

On context tabs below translation, you can use *Source* tab to provide feedback on source string or discuss it with other translators.

5.2.3 How can I use existing translations while translating?

Weblate provides you several ways to utilize existing translations while translating:

- You can use import functionality to load compendium as translations, suggestions or fuzzy translations. This is best approach for one time translation using compedium or similar translation database.
- You can setup *tmserver* with all databases you have and let Weblate use it. This is good for case when you want to use it for several times during translating.
- Another option is to translate all related projects in single Weblate instance, what will make it automatically pick up translation from other projects as well.

See Also:

Machine translation setup, *Machine translation*

5.2.4 Does Weblate update translation files besides translations?

Weblate tries to limit changes in translation files to minimum. For some file formats it might unfortunately lead to reformatting the file. If you want to keep the file formattted in your way, please use pre commit hook for that.

For monolingual files (see *Supported formats*) Weblate might add new translation units which are present in the *template* and not in actual translations. It does not however perform any automatic cleanup of stale strings as it might have unexpected outcome. If you want to do this, please install pre commit hook which will handle the cleanup according to your needs.

Weblate also will not try to update bilingual files in any way, so if you need `po` files being updated from `pot`, you need to do it on your own.

See Also:

Pre commit processing of translations

5.2.5 Where do language definition come from and how can I add own?

Basic set of language definitions is included within Weblate and Translate-toolkit. This covers more than 150 languages and includes information about used plural forms or text direction.

You are free to define own language in administrative interface, you just need to provide information about it.

5.2.6 Can Weblate highlight change in a fuzzy string?

Weblate supports this, however it needs the data to show the difference.

For Gettext PO files, you have to pass parameter `--previous` to **msgmerge** when updating PO files, for example:

```
msgmerge --previous -U po/cs.po po/phpmyadmin.pot
```

For monolingual translations, Weblate can find the previous string by ID, so it shows the differences automatically.

5.3 Troubleshooting

5.3.1 Requests sometimes fail with too many open files error

This happens sometimes when your Git repository grows too much and you have more of them. Compressing the Git repositories will improve this situation.

The easiest way to do this is to run:

```
# Go to GIT_ROOT directory
cd weblate/repos
# Compress all Git repositories
for d in */* ; do
    pushd $d
    git gc
    popd
done
```

See Also:

`GIT_ROOT`

5.3.2 Fulltext search is too slow

Depending on various conditions (frequency of updates, server restarts and other), fulltext index might get too fragmented over time. It is recommended to rebuild it from scratch time to time:

```
./manage.py rebuild_index --clean
```

See Also:

`rebuild_index`

5.3.3 I get "Lock Error" quite often while translating

This is usually caused by concurrent updates to fulltext index. In case you are running multi threaded server (eg. mod_wsgi), this happens quite often. For such setup it is recommended to enable `OFFLOAD_INDEXING`.

See Also:

Fulltext search

5.3.4 Rebuilding index has failed with "No space left on device"

Whoosh uses temporary directory to build indices. In case you have small /tmp (eg. using ramdisk), this might fail. Change used temporary directory by passing as `TEMP` variable:

```
TEMP=/path/to/big/temp ./manage.py rebuild_index --clean
```

See Also:

`rebuild_index`

5.4 Features

5.4.1 Does Weblate support other VCS than Git?

Weblate does not have native support for anything else than Git, however Git is versatile system, which allows plugging in remote helpers for other VCS as well.

At this time, helpers for Bazaar and Mercurial (hg) are available within Git source code, they might be also included in your Git package. If this is not the case, you can download them manually and put somewhere in your search path (for example `~/bin`). You also need to have installed appropriate version control programs.

Once you have these installed, you can use such remotes to specifi repository in Weblate.

To clone `gnuhello` project from Launchpad with Bazaar use:

```
bzr::lp:gnuhello
```

For `hello` repository from selenic.com with Mercurial use:

```
hg::http://selenic.com/repo/hello
```

Note: For native support of other VCS, Weblate requires distributed VCS and could be probably adjusted to work with anything else than Git, but somebody has to implement this support.

5.4.2 How does Weblate credit translators?

Every change made in Weblate is committed into VCS under translators name. This way every single change has proper authorship and you can track it down using standard VCS tools you use for code.

Additionally, when translation file format supports it, the file headers are updated to include translator name.

5.4.3 Why does Weblate force to have show all po files in single tree?

Weblate was designed in a way that every po file is represented as single subproject. This is beneficial for translators, that they know what they are actually translating. If you feel your project should be translated as one, consider merging these po files. It will make life easier even for translators not using Weblate.

Note: In case there will be big demand for this feature, it might be implemented in future versions, but it's definitely not a priority for now.

CHAPTER 6

Supported formats

Weblate supports any format understood by Translate-toolkit, however each format being slightly different, there might be some issues with not well tested formats.

See Also:

Supported formats in translate-toolkit

Weblate does support both monolingual and bilingual formats. Bilingual formats store two languages in single file - source and translation (typical examples is *GNU Gettext*, *XLIFF* or *Apple OS X strings*). On the other side, monolingual formats identify the string by ID and each language file contains only mapping of those to given language (typically *Android string resources*). Some file formats are used in both variants, see detailed description below.

For correct use of monolingual files, Weblate requires access to file containing complete list of strings to translate with their source - this file is called *Monolingual base language file* within Weblate, though the naming might vary in your application.

6.1 GNU Gettext

Most widely used format in translating free software. This was first format supported by Weblate and still has best support.

Weblate supports contextual information stored in the file, adjusting it's headers or linking to corresponding source files.

See Also:

https://en.wikipedia.org/wiki/Gettext, http://docs.translatehouse.org/projects/translate-toolkit/en/latest/formats/po.html

6.1.1 Monolingual Gettext

Some projects decide to use Gettext as monolingual formats - they code just IDs in their source code and the string needs to be translated to all languages, including English. Weblate does support this, though you have to choose explicitely this file format when importing resources into Weblate.

6.2 XLIFF

XML based format created to standardize translation files, but in the end it is one of many standards in this area.

XLIFF is usually used as bilingual.

See Also:

https://en.wikipedia.org/wiki/XLIFF, http://docs.translatehouse.org/projects/translate-toolkit/en/latest/formats/xliff.html

6.3 Java properties

Native Java format for translations.

Java properties are usually used as bilingual.

See Also:

https://en.wikipedia.org/wiki/.properties, http://docs.translatehouse.org/projects/translate-toolkit/en/latest/formats/properties.html

6.4 Qt Linguist .ts

Translation format used in Qt based applications.

Qt Linguist files are used as both bilingual and monolingual.

See Also:

http://qt-project.org/doc/qt-4.8/linguist-manual.html, http://docs.translatehouse.org/projects/translate-toolkit/en/latest/formats/ts.html

6.5 Android string resources

Android specific file format for translating applications.

Android string resources are monolingual, the *Monolingual base language file* file being stored in different location than others `res/values/strings.xml`.

See Also:

https://developer.android.com/guide/topics/resources/string-resource.html

---

Note: This format is not yet supported by Translate-toolkit (merge request is pending), but Weblate includes own support for it.

6.6 Apple OS X strings

Apple specific file format for translating applications, used for both OS X and iPhone/iPad application translations.

Apple OS X strings are usually used as bilingual.

See Also:

https://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPInternational/Articles/StringsFiles.html, http://docs.translatehouse.org/projects/translate-toolkit/en/latest/formats/strings.html

---

Note: Apple OS X strings are half broken in translate-toolkit 1.9.0 (it will generate corrupted files while saving), please use Git snapshot for handling these.

6.7 PHP files

PHP files can be processed directly, though currently Translate-toolkit has some problems writing them properly, so please double check that your files won't get corrupted.

PHP translations are usually monolingual, so it is recommended to specify base file with English strings.

Sample file which should work:

```
<?php

$string['foo'] = 'This is foo string';
```

See Also:

http://docs.translatehouse.org/projects/translate-toolkit/en/latest/formats/php.html

6.8 Others

As already mentioned, all Translate-toolkit formats are supported, but they did not (yet) receive deeper testing.

See Also:

Supported formats in translate-toolkit

CHAPTER 7

Weblate's Web API

7.1 Notification hooks

Notification hooks allow external applications to notify Weblate that Git repository has been updated.

GET /hooks/update/(string:project)/(string:subproject)/
Triggers update of a subproject (pulling from Git and scanning for translation changes).

GET /hooks/update/(string:project)/
Triggers update of all subprojects in a project (pulling from Git and scanning for translation changes).

POST /hooks/github/
Special hook for handling GitHub notifications and automatically updating matching subprojects.

Note: GitHub includes direct support for notifying Weblate, just enable Weblate service hook in repository settings and set URL to URL of your Weblate installation.

See Also:

http://help.github.com/post-receive-hooks/ `ENABLE_HOOKS`

POST /hooks/bitbucket/
Special hook for handling Bitbucket notifications and automatically updating matching subprojects.

See Also:

https://confluence.atlassian.com/display/BITBUCKET/POST+Service+Management
https://confluence.atlassian.com/display/BITBUCKET/Writing+Brokers+for+Bitbucket `ENABLE_HOOKS`

7.2 Exports

Weblate provides various exports to allow you further process the data.

GET /exports/stats/(string:project)/(string:subproject)/
Retrieves statistics for given subproject in JSON format.

You can get pretty-printed output by appending `?indent=1` to the request.

Example response:

```
[
    {
        "code": "cs",
        "failing": 0,
        "failing_percent": 0.0,
        "fuzzy": 0,
        "fuzzy_percent": 0.0,
        "last_author": "Michal \u010ciha\u0159",
        "last_change": "2012-03-28T15:07:38+00:00",
        "name": "Czech",
        "total": 436,
        "translated": 436,
        "translated_percent": 100.0,
        "url": "http://l10n.cihar.com/engage/weblate/cs/"
        "url_translate": "http://l10n.cihar.com/projects/weblate/master/cs/"
    },
    {
        "code": "nl",
        "failing": 21,
        "failing_percent": 4.8,
        "fuzzy": 11,
        "fuzzy_percent": 2.5,
        "last_author": null,
        "last_change": null,
        "name": "Dutch",
        "total": 436,
        "translated": 319,
        "translated_percent": 73.2,
        "url": "http://l10n.cihar.com/engage/weblate/nl/"
        "url_translate": "http://l10n.cihar.com/projects/weblate/master/nl/"
    },
    {
        "code": "el",
        "failing": 11,
        "failing_percent": 2.5,
        "fuzzy": 21,
        "fuzzy_percent": 4.8,
        "last_author": null,
        "last_change": null,
        "name": "Greek",
        "total": 436,
        "translated": 312,
        "translated_percent": 71.6,
        "url": "http://l10n.cihar.com/engage/weblate/el/"
        "url_translate": "http://l10n.cihar.com/projects/weblate/master/el/"
    },
]
```


Included data:

code language code

failing, **failing_percent** number and percentage of failing checks

fuzzy, **fuzzy_percent** number and percentage of fuzzy strings

last_author name of last author

last_change date of last change

name language name

total total number of strings

translated, translated_percet number and percentage of translated strings

url URL to access the translation (engagement URL)

url_translate URL to access the translation (real translation URL)

7.3 RSS feeds

Changes in translations are exported in RSS feeds.

GET /exports/rss/(string:project)/(string:subproject)/(string:language)/
Retrieves RSS feed with recent changes for a translation.

GET /exports/rss/(string:project)/(string:subproject)/
Retrieves RSS feed with recent changes for a subproject.

GET /exports/rss/(string:project)/
Retrieves RSS feed with recent changes for a project.

GET /exports/rss/language/(string:language)/
Retrieves RSS feed with recent changes for a language.

GET /exports/rss/
Retrieves RSS feed with recent changes for Weblate instance.

See Also:

https://en.wikipedia.org/wiki/RSS

CHAPTER 8

Changes

8.1 weblate 1.8

Released on November 7th 2013.

- Please check manual for upgrade instructions.
- Nicer listing of project summary.
- Better visible options for sharing.
- More control over anonymous users privileges.
- Supports login using third party services, check manual for more details.
- Users can login by email instead of username.
- Documentation improvements.
- Improved source strings review.
- Searching across all units.
- Better tracking of source strings.
- Captcha protection for registration.

8.2 weblate 1.7

Released on October 7th 2013.

- Please check manual for upgrade instructions.
- Support for checking Python brace format string.
- Per subproject customization of quality checks.
- Detailed per translation stats.
- Changed way of linking suggestions, checks and comments to units.
- Users can now add text to commit message.
- Support for subscribing on new language requests.

- Support for adding new translations.
- Widgets and charts are now rendered using Pillow instead of Pango + Cairo.
- Add status badge widget.
- Dropped invalid text direction check.
- Changes in dictionary are now logged in history.
- Performance improvements for translating view.

8.3 weblate 1.6

Released on July 25th 2013.

- Nicer error handling on registration.
- Browsing of changes.
- Fixed sorting of machine translation suggestions.
- Improved support for MyMemory machine translation.
- Added support for Amagama machine translation.
- Various optimizations on frequently used pages.
- Highlights searched phrase in search results.
- Support for automatic fixups while saving the message.
- Tracking of translation history and option to revert it.
- Added support for Google Translate API.
- Added support for managing SSH host keys.
- Various form validation improvements.
- Various quality checks improvements.
- Performance improvements for import.
- Added support for voting on suggestions.
- Cleanup of admin interface.

8.4 weblate 1.5

Released on April 16th 2013.

- Please check manual for upgrade instructions.
- Added public user pages.
- Better naming of plural forms.
- Added support for TBX export of glossary.
- Added support for Bitbucket notifications.
- Activity charts are now available for each translation, language or user.
- Extended options of import_project admin command.

- Compatible with Django 1.5.
- Avatars are now shown using libravatar.
- Added possibility to pretty print JSON export.
- Various performance improvements.
- Indicate failing checks or fuzzy strings in progress bars for projects or languages as well.
- Added support for custom pre-commit hooks and commiting additional files.
- Rewritten search for better performance and user experience.
- New interface for machine translations.
- Added support for monolingual po files.
- Extend amount of cached metadata to improve speed of various searches.
- Now shows word counts as well.

8.5 weblate 1.4

Released on January 23rd 2013.

- Fixed deleting of checks/comments on unit deletion.
- Added option to disable automatic propagation of translations.
- Added option to subscribe for merge failures.
- Correctly import on projects which needs custom ttkit loader.
- Added sitemaps to allow easier access by crawlers.
- Provide direct links to string in notification emails or feeds.
- Various improvements to admin interface.
- Provide hints for production setup in admin interface.
- Added per language widgets and engage page.
- Improved translation locking handling.
- Show code snippets for widgets in more variants.
- Indicate failing checks or fuzzy strings in progress bars.
- More options for formatting commit message.
- Fixed error handling with machine translation services.
- Improved automatic translation locking behaviour.
- Support for showing changes from previous source string.
- Added support for substring search.
- Various quality checks improvements.
- Support for per project ACL.
- Basic unit tests coverage.

8.6 weblate 1.3

Released on November 16th 2012.

- Compatibility with PostgreSQL database backend.
- Removes languages removed in upstream git repository.
- Improved quality checks processing.
- Added new checks (BB code, XML markup and newlines).
- Support for optional rebasing instead of merge.
- Possibility to relocate Weblate (eg. to run it under /weblate path).
- Support for manually choosing file type in case autodetection fails.
- Better support for Android resources.
- Support for generating SSH key from web interface.
- More visible data exports.
- New buttons to enter some special characters.
- Support for exporting dictionary.
- Support for locking down whole Weblate installation.
- Checks for source strings and support for source strings review.
- Support for user comments for both translations and source strings.
- Better changes log tracking.
- Changes can now be monitored using RSS.
- Improved support for RTL languages.

8.7 weblate 1.2

Released on August 14th 2012.

- Weblate now uses South for database migration, please check upgrade instructions if you are upgrading.
- Fixed minor issues with linked git repos.
- New introduction page for engaging people with translating using Weblate.
- Added widgets which can be used for promoting translation projects.
- Added option to reset repository to origin (for privileged users).
- Project or subproject can now be locked for translations.
- Possibility to disable some translations.
- Configurable options for adding new translations.
- Configuration of git commits per project.
- Simple antispam protection.
- Better layout of main page.
- Support for automatically pushing changes on every commit.

- Support for email notifications of translators.
- List only used languages in preferences.
- Improved handling of not known languages when importing project.
- Support for locking translation by translator.
- Optionally maintain Language-Team header in po file.
- Include some statistics in about page.
- Supports (and requires) django-registration 0.8.
- Caching of counted units with failing checks.
- Checking of requirements during setup.
- Documentation improvements.

8.8 weblate 1.1

Released on July 4th 2012.

- Improved several translations.
- Better validation while creating subproject.
- Added support for shared git repositories across subprojects.
- Do not necessary commit on every attempt to pull remote repo.
- Added support for offloading indexing.

8.9 weblate 1.0

Released on May 10th 2012.

- Improved validation while adding/saving subproject.
- Experimental support for Android resource files (needs patched ttkit).
- Updates from hooks are run in background.
- Improved installation instructions.
- Improved navigation in dictionary.

8.10 weblate 0.9

Released on April 18th 2012.

- Fixed import of unknown languages.
- Improved listing of nearby messages.
- Improved several checks.
- Documentation updates.
- Added definition for several more languages.

- Various code cleanups.
- Documentation improvements.
- Changed file layout.
- Update helper scripts to Django 1.4.
- Improved navigation while translating.
- Better handling of po file renames.
- Better validation while creating subproject.
- Integrated full setup into syncdb.
- Added list of recent changes to all translation pages.
- Check for not translated strings ignores format string only messages.

8.11 weblate 0.8

Released on April 3rd 2012.

- Replaced own full text search with Whoosh.
- Various fixes and improvements to checks.
- New command updatechecks.
- Lot of translation updates.
- Added dictionary for storing most frequently used terms.
- Added /admin/report/ for overview of repositories status.
- Machine translation services no longer block page loading.
- Management interface now contains also useful actions to update data.
- Records log of changes made by users.
- Ability to postpone commit to Git to generate less commits from single user.
- Possibility to browse failing checks.
- Automatic translation using already translated strings.
- New about page showing used versions.
- Django 1.4 compatibility.
- Ability to push changes to remote repo from web interface.
- Added review of translations done by others.

8.12 weblate 0.7

Released on February 16th 2012.

- Direct support for GitHub notifications.
- Added support for cleaning up orphaned checks and translations.
- Displays nearby strings while translating.

- Displays similar strings while translating.
- Improved searching for string.

8.13 weblate 0.6

Released on February 14th 2012.

- Added various checks for translated messages.
- Tunable access control.
- Improved handling of translations with new lines.
- Added client side sorting of tables.
- Please check upgrading instructions in case you are upgrading.

8.14 weblate 0.5

Released on February 12th 2012.

- **Support for machine translation using following online services:**
  - Apertium
  - Microsoft Translator
  - MyMemory
- Several new translations.
- Improved merging of upstream changes.
- Better handle concurrent git pull and translation.
- Propagating works for fuzzy changes as well.
- Propagating works also for file upload.
- Fixed file downloads while using FastCGI (and possibly others).

8.15 weblate 0.4

Released on February 8th 2012.

- Added usage guide to documentation.
- Fixed API hooks not to require CSRF protection.

8.16 weblate 0.3

Released on February 8th 2012.

- Better display of source for plural translations.
- New documentation in Sphinx format.

- Displays secondary languages while translating.
- Improved error page to give list of existing projects.
- New per language stats.

8.17 weblate 0.2

Released on February 7th 2012.

- Improved validation of several forms.
- Warn users on profile upgrade.
- Remember URL for login.
- Naming of text areas while entering plural forms.
- Automatic expanding of translation area.

8.18 weblate 0.1

Released on February 6th 2012.

- Initial release.

CHAPTER 9

Contributing

There are dozens od ways to contribute to Weblate. We welcome any help, be it coding help, graphics design, documentation or sponsorship.

9.1 Code and development

Weblate is being developed on GitHub <https://github.com/nijel/weblate>. You are welcome to fork the code and open pull requests. Patches in any other form are welcome as well. The code should follow PEP-8 coding guidelines.

We do write testsuite for our code, so please add testcases for any new functionality and verify that it works. You can see current test results on Travis <https://travis-ci.org/nijel/weblate> and coverage on Coveralls <https://coveralls.io/r/nijel/weblate>.

9.2 Translating

Weblate is being translated using Weblate on <http://l10n.cihar.com/>, feel free to join us in effort to make Weblate available in as many world languages as possible.

CHAPTER 10

License

CHAPTER 11

Indices and tables

- *genindex*
- *modindex*
- *search*