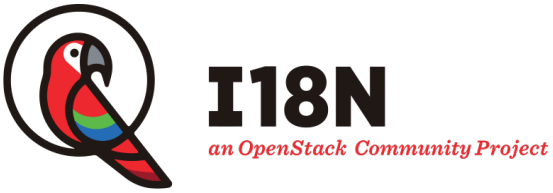
Internationalization Documentation

OpenStack Foundation

Mar 31, 2025

Contents



Internationalization (I18n) is essential to make OpenStack ubiquitous. The mission of OpenStack I18n team is to make OpenStack ubiquitously accessible to people of all language backgrounds, by enhancing OpenStack software internationalization, providing translation, maintaining a translation platform and managing translation process for better quality of outcomes.

This guide provides detailed instructions on the I18n contribution workflow and conventions to be considered by all contributors.

1 Contents

1.1 Contributing

The I18n team welcomes any kinds of contribution.

- Translations
- Translation tools
- Fixing I18n bugs

Translations

To help with translation, *become an official translator* and join your language team. Then go to translation website to start translation. You can find prioritized translation jobs as I18n team at the top of the page.

Your language team may have useful information to help our translation efforts including translation guidelines, priorities, communication tools and so on. It is worth visited. You can find your language team page at https://wiki.openstack.org/wiki/I18nTeam/team. If you have questions, you can contact your coordinators.

To report translation errors, see *Reporting translation bugs*.

Fixing I18n bugs

If you want to help fix I18n bugs of OpenStack, we can find them in the following places:

- openstack-i18n in Launchpad
- Horizon I18n bugs

Translation tools

Go to the *translation tools* page to understand the tools and scripts which support our translation platform.

If you want to help to report bugs, add more wishlists, and improve them, you can report and fix bugs at openstack-i18n in Launchpad. Mark bugs with a tag tools.

1.2 Reporting translation bugs

This page describes how end users can report translation bugs.

Finding and reporting translation bugs is an important step towards better quality of translations in OpenStack.

When you are using OpenStack (including the dashboard, API or CLI) or are reading translated documentations, if you find something wrong, something you cannot understand, something you cannot agree, or something to be improved, please consider reporting it.

Where can I report a bug?

OpenStack I18n team uses Launchpad for the bug tracking.

Translation bugs in all languages can be reported here.

How can I report a bug?

If you visit openstack-i18n launchpad, we can find **Report a bug** at the right-upper corner. Click to report a bug.

You can report a bug in your language (or English).

It is highly recommended to include the following information.

- Where you find a bug you are reporting?
  - For documentations, URL and a detail place would be great.
  - For the dashboard, panel or form name.
- What is wrong? What should be improved?
- **Make sure to include your English language name in a bug summary**. It is also suggested to write a bug summary in English. For example:

```
[japanese] Dashboard action in the Instance table is not translated
```

- (optional) Put your language name as bug tag.

Who can help you?

Everyone in the OpenStack community is glad to help you.

- Language translation team: you can ask in your language :-)
- OpenStack I18n team: See *Getting in touch*

1.3 Handling translation bugs

If you are a translator or a person involved in I18n effort, you may be interested in triaging and fixing translation bugs :)

The bug triaging process are following the general bug triaging process. If you want to help bug triaging tasks, join the OpenStack I18n bug team first.

Bugs in translated documents or dashboard are mainly classified into the following areas:

- translation bugs
- bugs in a source project
- bugs in tool chains

Translation bugs

If a bug reports translation errors of a certain language, it could be called **translation bug**. The translation bug should be fixed in the translation tool.

If you are a speaker of this language, you could help

- tag the bug with a language name

- confirm the bug or mark it as incomplete
- prioritize the bug

If you are a translator of this language, you could help

- assign the bug to you or a member of your language team, and then mark it as In Progress
- fix it by visiting a corresponding resource in Zanata and correcting translations.

> **Note**
>
> As a translator, if reported bugs turns out beyond a translation bug, it is better to ask I18n team members via the mailing list or the IRC channel. They can handle such bugs.

Bugs in a source project

The translatable strings are extracted from a source project. So some i18n bugs might be caused by bugs in the original strings/source codes of a source project which contains these translatable strings. Those kind of bugs should be fixed in the source project.

In most cases, a source project would be one of:

- OpenStack manuals
- Horizon (OpenStack dashboard)

You must determine whether bugs are **translation bugs** or **bugs in a source project**. Typical i18n bugs in the source project include:

- Original string or message is not correct.
- Missed translations. Missed translations might be caused by not extracting English strings from the source project, or be caused by real missed translations. If the translations are missed in two different languages, they are probably bugs in the source project. Or else, they are real missed translations.
- Bugs to report English strings are hard to translate in your language. For example, translators cannot control the order of words, or a plural form is not supported. Usually, this kind of bugs are in the original strings and cannot be fixed just by changing the strings and more work is needed.

If a reported bug turns out to be bugs in a source project, You could help

- tag it with the project name, for example horizon or docs.
- add the source project to Also affects project of this bug.
- confirm the bug
- prioritize the bug

Bugs in tool chains

If you encounter more complicated things including translation tool chains or something others, the most recommended way is to ask it in the I18n mailing list `openstack-i18n@lists.openstack.org`. Of course, you can file a bug against a related project directly.

1.4 Official OpenStack translator

Steps to become a OpenStack translator

Translation is another kind of important contribution to OpenStack community. If you want to become a official translator, you need to finish following steps:

1. Before you start contribution, youll have to agree to the contributor license agreement. (You can preview the full text of the OpenInfra Foundation Individual Contributor License Agreement first if you want.)

   Note

   If you want to become a translator only, simply speaking, you need to join The OpenInfra Foundation - Individual. You can see more details at Contributor Guide - Account Setup.

2. Register a user ID in Zanata
   - Go to Zanata server
   - Click Log in button.
   - If you dont have OpenInfra ID (previously, OpenStack ID), register one.
   - After you log in with OpenInfra ID, you will be requested to fill in your profile.

   Note

   You are encouraged to register with your business email, which will help your company to get the credit. If you dont want to, use your personal email will be OK too.

3. Request to join a translation team
   - Click Languages on the top, all languages will be listed.
   - Click the language you want to translate, the language page will be shown.
   - Click on the right, and select Request to join team.
   - Input a short introduction of yourself, including your name, as Additional information, then click Send message.

   Note

   Make sure to include a short introduction because it is the only information which language coordinators can use to determine your join request is valid or not.

4. When your request is approved, you will get an email notification.

   Note

   If your request is pending for long days, you can reach to your language coordinator through Zanata, to I18n people through IRC , or to I18n SIG Chair directly.

5. Now you can start your translation. You can actually become an OpenStack official translator by contributing translations. You can find *various ways of contributions*.

Active Contributor status in I18n project

The I18n team is one of official OpenStack SIGs (Special Interest Groups), so official translators who have contributed translations to the strings in code (e.g., Horizon) or documentations in official OpenStack projects or OpenStack SIGs in a specific period are regarded as AC (Active Contributor) of the I18n SIG. AC can vote for OpenStack TC (Technical Committee). For more detail on AC and TC, see OpenStack Technical Committee Charter.

Note

AC is a renamed term from ATC (Active Technical Contributor) which are the same. You can see more details on 2021-06-02 OpenStack ATC definition. This page also uses ATC term to mention old calculation and statistics.

As of now, AC of official translators are treated as extra ACs as we have no way to collect statistics automatically now. The list of extra ACs is maintained by the SIG Chair and is usually updated short before the deadline of extra ACs nomination in each release cycle. The deadline of extra ACs nomination can be checked in the release schedule page at http://releases.openstack.org/ (for example, http://releases.openstack.org/bobcat/schedule.html).

Translators who translate and review 300 and more words combinedly in the last six months until the deadline of extra ACs nomination are nominated as ACs, and the AC status of translators is valid for one year. Translation count and review count can be added up. The detail period is determined by the SIG Chair in each cycle. For Newton cycle, the six month period was from 2016-02-01 to 2016-07-31, and the ATC status expired on July 2017 for the translators and reviewers with no additional contributions.

Note

I18n SIG Chair updates the list using Zanata API and translator list. Detail statistics data is available *below*.

If you have a question, feel free to ask it to the SIG Chair or the i18n list.

Note that code or documentation contributors to openstack/i18n repository are acknowledged as AC automatically in the same way as for most OpenStack projects and/or SIGs.

AC members of I18n project

A current list all ACs is available at `i18n` part in https://opendev.org/openstack/governance/src/branch/master/reference/sigs-repos.yaml.

The statistics are calculated using a Python script powered by Zanata statistics API Translator list is maintained by translation_team.yaml stored in openstack/i18n git repository.

ATC members of I18n project

Before the transition to I18n SIG, I18n team was formed as one of official OpenStack projects and the ATC list was maintained by proposing a patch to projects.yaml on governance repository.

Note

On June 2021, ATC was renamed to AC. More information is available at one TC resolution.

ATC Statistics in past releases

Note

You can see detail member-level statistics data on HTML version of the page.

Train cycle

No detail data.

Stein cycle

- Period: 2018-07-10 to 2019-01-25
- Patch on governance repository: https://review.opendev.org/633398
- Foundation membership was validated by calling a REST API in https://openstackid-resources.openstack.org

Rocky cycle

- Period: 2018-01-11 to 2018-07-09
- Patch on governance repository: https://review.opendev.org/586751
- Foundation membership was validated by calling a REST API in https://openstackid-resources.openstack.org

Queens cycle

- Period: 2017-07-01 to 2018-01-10
- Patch on governance repository: https://review.opendev.org/532982
- Foundation membership was validated by calling a REST API in https://openstackid-resources.openstack.org

Pike cycle

- Period: 2017-01-06 to 2017-06-30
- Patch on governance repository: https://review.opendev.org/483452
- Foundation membership was validated by calling a REST API in https://openstackid-resources.openstack.org

Ocata cycle

- Period: 2016-08-01 to 2017-01-05
- Patch on governance repository: https://review.opendev.org/#/c/417569/ (diff)
- Note: the period is relative short because of release cycle change
  - More information: https://releases.openstack.org/ocata/schedule.html & https://ttx.re/splitting-out-design-summit.html
- [ianychoi] Note: only translated metric was considered to follow the description in I18n contributor guide. After more detail investigation on previous ATC criteria, review metrics were also considered. Fortunately, the three translators (jftalta - 117 translated & 2897 reviews, myamamot - 725 reviews, mucahit - 1241 reviews) are also regarded ATCs from Newton cycle :) - I keep it as record for the next ATC list changes.

Newton cycle

- Period: 2016-02-01 to 2016-07-31
- Patch on governance repository: https://review.opendev.org/#/c/351480/ (diff)

Mitaka cycle

- Period: 2015-08-01 to 2016-01-30
- Patch on governance repository: https://review.opendev.org/#/c/281145/ (diff)
- This following statistics data is calculated using up-to-date translation_team.yaml (date: Jan 15, 2017).
- When proposing extra ATCs at that time, some translators were not included in translation_team.yaml file. It seems that 1) new translators were joined and the statistics was calculated but the file was already created, or 2) there might be some lack of communication with language coordinators, since I18n encouraged each language coordinator to update this file.
- [ianychoi] Although one year was already passed (as of now: Jan 15, 2017), I really would like to say those translators also contributed translations with I18n team members. For acknowledgement purpose, I write their Zanata ID, name, and language team in here :
  - Zbynk Schwarz (id: tsbook) - Czech
  - Rob Cresswell (id: robcresswell) - English (United Kingdom)
  - Heleno Jimenez de la Cruz (id: heleno_jimenez) - Spanish (Mexico)
  - Jori Kuusinen (id: nuyori) - Finnish (Finland)
  - Masaki Matsushita (id: mmasaki) - Japanese
  - Amandeep Singh Saini (jimidar) - Punjabi (India)
  - ukasz Jernas (id: deejay1) - Polish (Poland)

Liberty cycle

- Period: from 2014-11-01 to 2015-07-16
- Patch on governance repository: https://review.opendev.org/#/c/213989/ (diff)
- exported the translators contribution statistics from Transifex since 2014-11-01 to 2015-07-16
- ATC candidates are translators who has translated more than 200 words and reported their e-mail and name to language coordinators, and also signed ICLA.
- More information on mailing list: http://lists.openstack.org/pipermail/openstack-i18n/2015-July/001220.html
- Statistics are available through: https://docs.google.com/spreadsheets/d/1YpDJU_uNA4I5fzFG69T6L9gpFsy5yNtA9a-lSxnqeAY/edit#gid=1366189722

1.5 Language Team

OpenStack I18n team is a team to coordinate all language translations and cooperation with the I18n team and the development teams. I18n team usually has a working plan.

Although each language team could follow the I18n team working plan, your team may also decide your own work items and priorities as a language team.

Each language team may have useful information to help our translation efforts including translation guidelines, priorities, communication tools and so on. It is worth visited. You can find your language team page at https://wiki.openstack.org/wiki/I18nTeam/team. If you have questions, you can contact your coordinators.

If you cannot find your local team, you can request to *create a local translation team*.

Roles in a Language Team

There are three different roles in a translation team:

Member (translator)
: A person who can submit translations.

Reviewer
: A person who can proofread translations and mark them as reviewed (approved) or rejected.

Coordinator
: A privileged member in a translation team who can help in team management tasks, such as approving new members and reviewing contributions to that language. We also have task coordinators for specific tasks.

Managing a Language Team

This section tries to share best practices on how team coordinators manages their language team. Feel free to add useful information!

- When a new member request is received, a coordinator approves (or rarely rejects) the request. At the moment, Zanata provides no way to communicate with language team members, so it is better to inform the new member of ways usually used in the language team.
- When a translation period towards a new OpenStack release approaches, it is recommended to coordinate and prioritize translation efforts.

- Setting up a communication channel among team members is recommended. It encourage team communications to discuss various translation related topics.
- It is a good idea to organize glossaries and initiate related discussion.
-

Of course there is no need that team coordinators do all items. You can collaborate your team members and share ideas.

Creating a Language Team

If you want to create a language team and you want to be a coordinator, please follow below steps:

1. Follow *the step* to register an ID in the translation website.
2. Write an email to openstack-i18n@lists.openstack.org to introduce yourself and the language team you want to create, together with your ID on Zanata.
3. When your request is approved, add an item in Local Translation Teams in the wiki page, create a wiki page under page I18nTeam (for example, I18nTeam/team/zh_cn) to introduce your team, with the following information at least: the contact information, the web page, and the work items.

Then, your locale translation team is created successfully. You can enroll more people to join and start your translation work.

Keep in mind to work with your local user group and the OpenStack Ambassadors to raise awareness and gather members :)

1.6 I18n SIG Chair Guide

This chapter describes tasks for I18n SIG Chair and gives some useful hints. If you want to be I18n SIG Chair or you new in this role, please read carefully. For all other is this chapter informal.

Note

Since I18n team previously followed as a form of official projects, some of the below description may assume detail tasks as PTL responsibility. See Project Team Guide which is a reference for PTLs for more information, and I18n team now follows as a form of working group defined as SIGs (Special Interest Groups).

Cross Project Liaisons

I18n team participates in various cross project liaisons like Release Management, Documentation and Infrastructure tasks. Many other teams are liaisons in I18n. Visit the Cross Project Liaisons Wiki Page and designate a person for the I18n team. Note that such liaison roles do not have to be I18n SIG Chair. Active I18n cores are highly encouraged.

Project Goals And Translation Plan

In the beginning of each cycle it is a good idea to think about goals for the project in the next months and establish a translation plan. Goals are maybe still left from the last cycle and are to be reviewed. New goals are to be defined on PTG or an equivalent event. Orient the team to the OpenStack-wide goals.

The translation plan is announced on Zanata and contains the projects with the highest priority in translation. Usually these are all user-visible projects, like Horizon.

I18n Core Team

The SIG Chair is supported in the work by the I18n core team. The person designates such kind of project team members and reviews the list from time to time. The usual work of the core team is described in the Project Team Guide. Of course, core team member be can also proposed by the project team.

Launchpad I18n Core Team

As with I18n Core Team, the same things apply here. Launchpad I18n Core Team focuses on determining the importance of a translation and/or I18n bug and triaging bug priorities. You can find the member list on Launchpad I18n Core Member list.

Release Management

The work for SIG Chair and Zanata administrator is described in chapter *Team activities with release*. This covers also questions about string freezes and work with stable branches.

Extra-ACs Deadline

Each cycle has a date set for Extra-ACs, e.g. Queens Cycle. Maintenance on I18n site is described in chapter *Project maintenance*. All the OpenStack members can propose extra ACs, but I18n SIG Chair is highly encouraged to report the list in each cycle. Here are some useful proposals as example:

- https://review.opendev.org/#/c/488226/
- https://review.opendev.org/#/c/483452/
- https://review.opendev.org/#/c/451625/

This Script in I18n repo collects all users and their activities.

Daily Work

Translation Job Control

Translated strings are automatically synced between translation server and Openstack infrastructure. The procedure is robust, but sometimes something can go wrong. For this reason there is a section *Monitoring translation jobs status* in the infrastructure chapter.

You can check on Gerrit, if the translated strings are imported by the project teams. Core reviewers in each repository are strong encouraged to approve translation sync patches but do not be sad if the translations are not accepted. Zanata Sync jobs are repeated every day until they are merged.

Open reviews I18n repository

Check from time to time open reviews on I18n repo. In addition to the PTL, the core reviewers are responsible.

Launchpad bugs & blueprints

Translation bugs are tracked on Launchpad. Often only the language teams are able to handle translation bugs, e.g. wrong translation words or senses. PTL fits thereon or designates bug triage liaison.

I18n blueprints are listed also on Launchpad. In normal case a blueprint has an assignee and describes a larger course of a process.

I18n Mailing List Management

In addition to IRC, I18n team communication takes place via a mailing list. The Mailing List Administrator is watching new subscribers, not allowed posts, and all the other things that come with the operation of such a list.

1.7 How to check translations

It is important to validate your translations by applying them in a real situation where they are rendered and visualized. This page describes how to check your translations.

Note

`translation check` refers to build OpenStack artifacts with translated strings and check how the translated strings are shown on an actual screen as part of the translation process.

Documentation

Translated documents are available at the OpenStack Documentation site. It is updated daily. Most contents are linked from:

- https://docs.openstack.org/<lang> contains released documents. Follow Languages in https://docs.openstack.org/.

The documents are maintained in the `doc` directory using reStructuredText and built by Sphinx. Translated strings are stored as Gettext PO file format in `locale/<lang>/LC_MESSAGES` directory.

Documents on openstack-manuals

For OpenStack documentation, openstack-manuals git repository hosts essential OpenStack documents such as Installation Guide, and Virtual Machine Image Guide.

To build a translated document on this repository, you need to update the file `doc-tools-check-languages.conf` in each repository, and add an entry to `BOOKS` like `["ja"]="install-guide"`.

For a document in a stable branch, such as the installation guide for Liberty, you need to update the file `doc-tools-check-languages.conf` in the target stable branch directly.

You can check a generated document for a specified branch on `https://docs.openstack.org/<branch>/<language>/<document>`. For example, the link of Japanese Ubuntu Installation Guide for Liberty is https://docs.openstack.org/liberty/ja/install-guide-ubuntu/.

To add a link to a generated document, you need to update the file `www/<lang>/index.html` in the `master` branch of the `openstack-manuals` repository. Note that the web pages are published from `master` branch, which contains the pages for all releases, such as Liberty. Therefore, you dont need to update the file `www/<lang>/index.html` in the stable branch.

You can also check:

- build status for publishing on Zuul
- checkbuild with drafts on Zuul

OpenStack project documentation

Currently, we support translations for small set of OpenStack project documentations like Horizon, OpenStack-Ansible, and OpenStack-Helm upon requests and available bandwidth. Top-level directory structure on `doc/source/` follows with Documentation Contributor Guide - Project guide setup and the corresponding generated documents for `master` branch are available at `https://docs.openstack.org/<project>/latest/<language>/<document>` URL. For project team documents in stable branch, you can check the documents with `https://docs.openstack.org/<project>/<branch>/<language>/<document>` URL.

Here are sample document links as examples:

- Korean OpenStack-Helm Installation Guide (no branch)
- German OpenStack-Ansible User Guide for 2023.1

OpenStack Dashboard

Running OpenStack-Ansible

OpenStack-Ansible (OSA) provides Ansible playbooks and roles for the deployment and configuration of an OpenStack environment. As part of the project a feature named Translation Check Site is developed. An OSA instance will fetch translation strings from translation platform, compile and serve these strings in Horizon. You need a machine with two or four CPU cores, at least 8 GB memory and 70 GB disk to run OSA.

```
$ BRANCH=master
$ git clone -b ${BRANCH} https://github.com/openstack/openstack-ansible /opt/
↪openstack-ansible
$ cd /opt/openstack-ansible
$ ./scripts/gate-check-commit.sh translations
```

You can set the components of your AIO installation in `tests/vars/bootstrap-aio-vars.yml`. Dependly on your environment the installation takes 1-2 hours. For more details on the AIO configuration, please see OSA AIO documentation.

To fetch translated files regularly, execute this command manually or as a cron:

```
$ cd /opt/openstack-ansible/playbooks; \
  openstack-ansible os-horizon-install.yml \
  -e horizon_translations_update=True \
  -e horizon_translations_project_version=master \
  --tags "horizon-config"
```

Running DevStack

Another convenient way is to check dashboard translations is to run DevStack in your local environment. To run DevStack, you need to prepare `local.conf` file, but no worries. Several `local.conf` files are shared on the Internet and a minimum example is shown below. From our experience, you need a machine with two or four CPU cores, 8 GB memory and 20 GB disk to run DevStack comfortably. If you enable just major OpenStack projects, the machine requirement would be much smaller like 2~4GB memory.

```
$ BRANCH=master
$ git clone https://opendev.org/openstack/devstack.git
```

(continues on next page)

(continued from previous page)

```
$ cd devstack
$ git checkout $BRANCH
<prepare local.conf>
$ ./stack.sh
<wait and wait... it takes 20 or 30 minutes>
```

Replace `$BRANCH` with an appropriate branch such as `master`, `stable/newton` or `stable/mitaka`.

The following is an example of `local.conf` for Newton release which runs core components (keystone, nova, glance, neutron, cinder), horizon, swift and heat. The components which the main horizon code supports are chosen.

```
[[local|localrc]]
BRANCH=stable/newton
# NOTE: We need to specify a branch explicitly until DevStack stable branch
# is prepared. At now, swift has no stable/newton branch.
CINDER_BRANCH=$BRANCH
GLANCE_BRANCH=$BRANCH
HEAT_BRANCH=$BRANCH
HORIZON_BRANCH=$BRANCH
KEYSTONE_BRANCH=$BRANCH
NEUTRON_BRANCH=$BRANCH
NOVA_BRANCH=$BRANCH
SWIFT_BRANCH=master

# When OS_CLOUD envvar is set, DevStack will be confused.
unset OS_CLOUD

# Ensure to fetch the latest repository when rerunning DevStack
RECLONE=True

# Translation check site usually does not use tempest.
disable_service tempest

enable_service heat h-api h-api-cfn h-api-cw h-eng
enable_service s-proxy s-object s-container s-account

enable_plugin neutron https://git.openstack.org/openstack/neutron $BRANCH
enable_service q-qos

enable_plugin neutron-vpnaas https://git.openstack.org/openstack/neutron-
↪vpnaas $BRANCH

LOGFILE=$DEST/logs/devstack.log
SCREEN_HARDSTATUS="%{= rw} %H %{= wk} %L=%-w%{= bw}%30L> %n%f %t*%{= wk}%+Lw%-
↪17< %-=%{= gk} %y/%m    /%d %c"
LOGDAYS=2
IDENTITY_API_VERSION=3

ADMIN_PASSWORD=change_me
```

(continues on next page)

(continued from previous page)

```
MYSQL_PASSWORD=change_me
RABBIT_PASSWORD=change_me
SERVICE_PASSWORD=change_me
```

Import latest translations

Translations are being imported into a project repository daily, so in most cases you do not need to pull translations from Zanata manually. What you need is to pull the latest horizon code.

If you have a machine running DevStack, there are two ways.

One way is to update the horizon code only. The following shell script fetches the latest horizon code, compiles translation message catalogs and reloads the apache httpd server. Replace `$BRANCH` with an appropriate branch such as `master`, `stable/newton` or `stable/mitaka`.

```
#!/bin/bash

# Target branch: master, stable/pike, ...
BRANCH=stable/pike

cd /opt/stack/horizon

# Remove stale compiled python files
find horizon -name '*.pyc' | xargs rm
find openstack_dashboard -name '*.pyc' | xargs rm

# Fetch the latest code from git
git checkout $BRANCH
git remote update origin
git merge origin/$BRANCH

python manage.py compilemessages
python manage.py collectstatic --noinput
python manage.py compress --force
sudo service apache2 reload
```

The other way is to rerun DevStack. Ensure to include `RECLONE=True` in your `local.conf` before running `stack.sh` again so that DevStack retrieve the latest codes of horizon and other projects.

```
$ cd devstack
$ ./unstack.sh
<Ensure RECLONE=True in your local.conf>
$ ./stack.sh
<It takes 10 or 15 minutes>
```

CLI (command line interface)

TBD

Server projects

TBD

1.8 Translation tips

This page collects small tips and tricks around translations.

If you have any, dont hesitate to share it :)

Where a string is used in source code?

It is important to understand how an original string is used in the source code with contextual knowledge for better quality of translations. Unfortunately Zanata translation interface does not show where a target string is used in a source code correspondingly.

You can find the location in a source code by checking the POT file.

The POT file is found under `http://tarballs.openstack.org/translation-source/<Zanata-project-name>/<Zanata-project-version>/<path>/<resource>.pot`, where:

- `<Zanata-project-name>` is Zanata project name,
- `<Zanata-project-version>` is Zanata project version such as `master` or `stable-mitaka`,
- `<path>` is a path of a document such as `nova/locale`, `openstack_dashboard/locale` or `releasenotes/source/locale`,
- `<resource>` is a document name in Zanata.

The easiest way would be to open http://tarballs.openstack.org/translation-source and then follow `<Zanata-project-name>`, `<Zanata-version>` and corresponding links.

Open a POT file you find and search a string you are interested in. Then you can find an entry like:

```
#: trove_dashboard/content/database_backups/views.py:100
#, python-format
msgid "Unable to retrieve details for parent backup: %s"
msgstr ""
```

Once you find location(s) of a string you would like to check, you can check more detail context where the string is used by looking at the code in the git repository. Zanata project and git repository are one-to-one relationship. If Zanata project name is `<Zanata-project-name>`, the corresponding git repository location is `https://opendev.org/openstack/<Zanata-project-name>/`. For example, if you are translating `horizon`, the git repository is found at https://opendev.org/openstack/horizon/.

Note

POT files are no longer stored in git repositories. The change was made at the beginning of Newton development cycle[1].

[1] http://lists.openstack.org/pipermail/openstack-dev/2016-May/094215.html

Where a string is used in Dashboard UI?

During translating a dashboard related project, you may want to know where is this string used in the dashboard?.

You can estimate where a string is used from a location in a source code. (See the previous entry on how to find a location.)

dashboards/admin/aggregates/views.py:104
: This means Aggregates menu in Admin group in the left side menu. In Horizon code, the second level corresponds to a dashboard group like Project or Admin and the third level corresponds to a panel like Aggregates (in this example), Instances or Networks.

dashboards/project/loadbalancers/workflows.py:44
: When a filename is `workflows.py` or `forms.py`, it means it is used in a form for creating or editing something.

dashboards/project/containers/templates/containers/_update.html:21
: A template is a skeleton of static HTML files. In this case, the string is perhaps used in Update form in Container menu.

1.9 Glossary Management

Zanata provides suggestions for translation based on a glossary of your language. The glossary is maintained in the i18n repository.

Glossary of your language

A glossary file of your language is found at `glossary/locale/<lang>/LC_MESSAGES/glossary.po` in the i18n repository. The file is a usual PO file.

When you want to update a glossary of your language, edit a corresponding glossary file using your favorite editor and propose a change to the Gerrit review system.

Once your change is merged, the updated glossary will be uploaded to Zanata. Note that the upload to Zanata is a **manual** process now. Only the Zanata administrator (usually the I18n SIG Chair and some limited folks like core reviewers) can upload a glossary. If your glossary is not uploaded, contact the I18n team or the SIG Chair via the i18n mailing list.

> **Note**
>
> Discuss and review glossary changes in your language team before you propose changes to the Gerrit review system. Reviewers in the Gerrit review system cannot understand your language in most cases, so they can only check syntax or conventions.

Master glossary

When you add a new entry to the glossary, it is highly recommended to add the entry to the master glossary `glossary/locale/glossary.pot`. By doing this, all language teams can share the glossary.

The master glossary is an usual POT file. You can edit it in your favorite editor as usual.

After you edit the master glossary, ensure to reflect the change to all language glossary files. To do this, run the following command.

```
tox -e venv python setup.py update_catalog
```

Finally, propose a patch including all of the above changes to the Gerrit review system.

1.10 Translation Statistics

There are several ways to know your translation activity.

Zanata

Zanata provides ways to know your activity on Zanata.

- Dashboard page shows your recent activity statistics this month and detail activity.
- Profile page provides more nice statistics view. It shows your statistics in this month and last month.

Note that you need to log into Zanata to see your activity in the above pages.

Stackalytics

Stackalytics is a popular web site which allows us to know various statistics related to OpenStack. It supports translation statistics :)

Visit Stackalytics and choose `Translations` as `Metric` dropdown menu at the upper-right.

FAQ: I cannot find my name in Stackalytics

There are cases where you cannot see your translation statistics in Stackalytics even after you translate strings in Zanata.

Here is the check list for such case:

- Is your Zanata ID included in the translator list?
- Is your Zanata ID different from your launchpad ID?

If your Zanata ID is not included in the translator list, you need to update the list to include your Zanata ID. Contact your language coordinator, email the i18n mailing list. You can also submit a patch to update the list by yourself (For detail, see *Sync the translator list with Zanata*).

If your Zanata ID is different from your launchpad ID, Stackalytics will not find your translation statistics. You need to let Stackalytics know your ID mappings. To do this, you need to add your user data into `detault_data.json` in the Stackalytics repository.

If you are lucky to use a same name for launchpad and Zanata IDs, you do not need to do the above. Stackalytics will find your statistics automatically.

1.11 Tools

This page covers various operations around i18n activities.

Zanata CLI

OpenStack uses Zanata as a translation platform. While most operations around the translation platform are automated, if you want to communicate with the translation platform manually, you can use Zanata CLI.

User configuration

You need to create a configuration file in `$HOME/.config/zanata.ini` that contains user-specific configuration. For information on how to create a configuration file, see Zanata CLI configuration.

Project configuration

To communicate with the translation platform, you need to prepare a project configuration file named `zanata.xml` in the top directory of a project you are interested in. OpenStack projects does not contain `zanata.xml` in their git repositories, so you need to create it manually.

The following is an example of `zanata.xml`. In most cases, what you need to edit are **project** and **project-version**.

```
<config xmlns="http://zanata.org/namespace/config/">
  <url>https://translate.openstack.org/</url>
  <project>horizon</project>
  <project-version>master</project-version>
  <project-type>gettext</project-type>
  <src-dir>.</src-dir>
  <trans-dir>.</trans-dir>
  <rules>
    <rule pattern="**/*.pot">{path}/{locale_with_underscore}/LC_MESSAGES/
↪{filename}.po</rule>
  </rules>
  <excludes>.tox/**</excludes>
</config>
```

Pull translations from Zanata

To download translations from Zanata, run the following command after going into a project directory. You are usually interested in only a few of languages, so `--locales` option would be useful. For more options, see the output of `zanata pull --help`.

```
$ zanata-cli pull --locales ja,ko-KR,zh-CN
```

Handling documentation projects

Note

This is written about openstack-manuals project. As of the end of Pike development cycle, the document migration community-wide effort is being done. The process documented here might be changed in near future.

OpenStack documents are using RST format. The steps to translate RST documents include:

- Slicing: generate PO templates from RST documents
- Uploading: Upload the translation resources to Zanata
- Translating: manage the translation in Zanata, including the translation memory and glossary management

- Downloading: Download the translated results by automation scripts.
- Building: Build HTML from RST documents and the translated results.

Sphinx is a tool to translate RST source files to various output formats, including POT and HTML. You need to install Sphinx before you go to below steps. Almost all projects have `test-requirements.txt` in their repositories and you can check the required version of Sphinx by checking this file.

```
$ pip install Sphinx
```

Or, more convenient way would be:

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

Slicing

We use sphinx-build to translate RST files to POT files. Because we want to have a single POT file per document, we use msgcat to merge those POTs after sphinx-build.

```
$ sphinx-build -b gettext doc/[docname]/source/ doc/[docname]/source/locale/
$ msgcat doc/[docname]/source/locale/*.pot > doc/[docname]/source/locale/
↪[docname].pot
```

Uploading

We use *Zanata CLI* to upload the POT file to the translate platform.

Downloading

We use *Zanata CLI* to download the translated PO files from the translation platform.

Building

Before use sphinx-build to build HTML file, we need to feed the translations from the single PO file into those small PO files. For example:

```
$ msgmerge -o doc/[docname]/source/locale/zh_CN/LC_MESSAGES/A.po \
    doc/[docname]/source/locale/zh_CN/LC_MESSAGES/[docname].po \
    doc/[docname]/source/locale/A.pot
```

Then, for every PO file, we should execute the following command to build into MO file:

```
$ msgfmt doc/[docname]/source/locale/zh_CN/LC_MESSAGES/A.po \
   -o doc/[docname]/source/locale/zh_CN/LC_MESSAGES/A.mo
```

Finally, we could generate HTML files by

```
$ sphinx-build -D "language='zh_CN' doc/[docname]/source/ \
    doc/[docname]/build/html
```

Handling python projects

For most of the Python projects, the preferred tools for I18N are gettext and babel. The gettext module provides internationalization (I18N) and localization (L10N) services for your Python modules and applications. Babel are a collection of tools for internationalizing Python applications.

Extracting

You can extract the messages in code to PO template (POT) with pybabel, where **PROJECT** is a project name like `nova` and **VERSION** is a version number. Note that you can omit `--project` and `--version` options if you just use them locally as they are just used in the POT file header.

```
$ pybabel extract \
    --add-comments Translators: \
    -k "_C:1c,2" -k "_P:1,2" \
    --project=${PROJECT} --version=${VERSION} \
    -o ${modulename}/locale/${modulename}.pot \
    ${modulename}
```

For example, in case of nova,

```
$ pybabel extract \
    --add-comments Translators: \
    -k "_C:1c,2" -k "_P:1,2" \
    --project=nova --version=${VERSION} \
    -o nova/locale/nova.pot nova/
```

Uploading

For each Python project in OpenStack, there is an automation job to extract the messages , generate PO template and upload to Zanata, which is triggered by the commit event. See *here*.

Downloading

For each Python project in OpenStack, there is an automation job daily to download the translations in PO file to the locale folder under the source folder of each project. See *here*. It will generate a review request in Gerrit. After *review*, the translation in PO file will be merged.

Using translations

To display translated messages in python server projects, you need to compile message catalogs and also need to configure your server services following instructions described at oslo.i18n documentation.

Handling horizon projects

For horizon related projects, Django, a framework which horizon is built on, provides integrated translation support.

Note

Unlike documentations and python projects, horizon and plugins use `zh-hans` and `zh-hant` for Chinese locales instead of `zh-cn` and `zh-tw` respectively since Wallaby release. This follows the Django

recommendation which happened in Django 1.7. The details are found in the mailing list post.

Extracting

horizon provides a command to extract the messages in code to PO template (POT). Run the following command in your repository.

```
$ python manage.py extract_messages -m ${MODULE_NAME}
```

where **MODULE_NAME** is a python module name of horizon or its plugin.

For example, in case of manila-ui,

```
$ python manage.py extract_messages -m manila_ui
```

The above command is a wrapper for pybabel and the translation job uses pybabel directly.

Uploading

For each horizon related project in OpenStack, there is an automation job to extract the messages , generate PO template and upload to Zanata, which is triggered by the commit event. See *here*.

Downloading

For each horizon related project in OpenStack, there is an automation job daily to download the translations in PO file to the locale folder under the source folder of each project. See *here*. It will generate a review request in Gerrit. After *review*, the translation in PO file will be merged.

Note

As noted above, in Wallaby or later releases, `zh-hans` and `zh-hant` are used for Chinese locales. On the other hand, `zh-cn` and `zh-tw` continues to be used in Zanata. When the job downloads translations from Zanata, the job stores translations for `zh-cn` and `zh-tw` to `zh-hans` and `zh-hant` directories under `locale`.

Using translations

To display translated messages in OpenStack dashboard, you need to compile message catalogs. Django picks up translated messages automatically once they are compiled. The following command compiles messages in your project.

```
$ python manage.py compilemessages
```

Project maintenance

Note

The scripts below depend on several python modules. To install these dependencies, run `pip install -e requirements.txt`.

More convenient way is to use **tox** like `tox -e venv -- python <script-name>`.

tox is available on PyPI and also available in various Linux distribution. `pip install tox` or `apt-get install python-tox` (in case of Ubuntu) installs **tox**.

Sync the translator list with Zanata

The I18n project maintains a list of language teams and their members. The list is used by Stackalytics to gather translation statistics (See *Stackalytics* for detail). It is also used by the scripts below.

The filename of the list is `tools/translation_team.yaml`.

This list is a cache of information on Zanata, and we need to keep it synced with Zanata.

To sync the translator list, run the following command:

```
tox -e zanata-users-sync
```

The above run the following internally:

```
python tools/zanata/zanata_users.py --output-file tools/zanata/translation_
↪team.yaml
```

Retrieve translation statistics

AC status in I18n project is determined based on translation statistics in a specific period.

The script `tools/zanata/zanata_stats.py` helps retrieving translation statistics from Zanata.

To run the script:

```
tox -e venv -- python ./tools/zanata/zanata_stats.py <options>
```

`--help` option shows the detail usage.

Extract Zanata user information

At the moment, the I18n SIG Chair needs to maintain the AC list of the I18n project manually around the end of each release cycle. This requires name and e-mail address of individual translators.

The script `tools/zanata/zanata_userinfo.py` helps this. It generates a CSV file by reading a YAML file which contains the list of translators (e.g., `translation_team.yaml`) with user name and e-mail addresses by interacting with Zanata API.

Note

This script requires Zanata admin privilege.

1.12 I18n team meeting

The I18n team holds weekly Team Meetings on Wednesdays at alternating weekly basis in #openstack-i18n IRC channel. To download ICS file, please visit OpenDev - IRC channels and meetings page.

- One week: 13:30 UTC, Wednesday

- The other week: 23:15 UTC, Wednesday

> **Note**
>
> Odd and even values are calculated based on ISO week numbers.

Meeting agenda

Everybody can propose agenda topics on I18nTeamMeeting Wiki Page. If there are no topics proposed, the meeting might be canceled for that week.

Usual topics

- Review of action items from the previous meeting
- Translation activities with plans from language teams
- Launchpad bugs, Gerrit reviews, and broken Zuul jobs
- Open discussion

Handy links (always sort of on the agenda)

- Translation plan
- Translation statistics
- Bug lists
- Open reviews in openstack/i18n
- Broken Translation Jobs on Zuul

Previous meetings

You can refer to previous meeting logs with their notes.

IRC web client

If you are not familiar with IRC, please read Contributor Guide. You can simply use the web client for OFTC to come to #openstack-i18n channel.

IRC meeting commands

This is a useful commands for the IRC meeting.

```
#startmeeting openstack_i18n_meeting
#topic open discussions
#info useful information
#link http://contents
#endmeeting
```

1.13 Translation infrastructure

A series of tasks in OpenStack infrastructure is used to manage translation changes in Zanata. Without running the tasks, translation changes will not be reflected into OpenStack projects. This page explains how the infrastructure tasks run actual scripts as Zuul jobs and monitor the job status.

Translation jobs

We have two types of Zuul jobs for translations: syncing source strings into Zanata with the latest repositories and pushing translations from Zanata into the repositories. The first job is for Zanata-side updates. Up-to-date source strings to be translated are compared and updated between OpenStack project repositories and Zanata. If source texts in OpenStack project repositories are changed, then change sets are pushed into Zanata so translators deal with up-to-date source strings. On the other hand, the second job is aimed to reflect changes in translated strings in Zanata (after translators do translation activities) into corresponding OpenStack project repositories. The job will propose changes as *Translation Import* Gerrit patches.

Update jobs for Zanata start after patches are merged on OpenStack project repositories, and Zuul starts to run tasks everyday at **6:00 UTC** for the updates on OpenStack project repositories.

Note that not all translation changes are the target for translation jobs. The goal is to have consistent translated programs, UIs, and documentation. There is not much sense if only a few lines are translated. The team has decided that files that have at least 75 percent of messages translated will be in the git repositories.

To not have too much churn and last minute string fixes lead to files get removed, there is also a lower threshold for releases of **66 percent** of messages translated as policy - which is only manually enforced.

The OpenStack infra scripts executed by tasks currently download new files that are at least **75 percent** translated and if files grow over time but do not get new translations (or strings change too much), they will be removed again automatically from the project with a lower threshold of currently **40 percent**.

Monitoring translation jobs status

The Zuul Builds Page provides us a convenient way to check the translation job status.

- Post jobs - Syncing to Zanata
- Periodic jobs: Syncing into repos

Translation infrastructure tasks and scripts

Translation infrastructure tasks are stored and managed in openstack/project-config repository. The translation infrastructure scripts are stored and managed in openstack/openstack-zuul-jobs repository.

- upstream-translation-update.yaml
  - Implements the first Zuul job (Syncing to Zanata) by executing upstream_translation_update.sh
- propose-translation-update.yaml
  - Carries out the second Zuul job (Syncing into repos) by executing propose_translation_update.sh
  - This job renames Chinese locales in Zanata (`zh-cn` and `zh-tw`) to `zh-hans` and `zh-hant` for horizon related projects. For other projects and documentations (including docs in horizon related projects), `zh-cn` and `zh-tw` are used as-is. For more details, see *Handling horizon projects*.
- common_translation_update.sh
  - Common code used by **propose_translation_update.sh** and **upstream_translation_update.sh**

- create-zanata-xml.py
  - Python script to setup projects for Zanata
- releasenotes/pre.yaml
- releasenotes/run.yaml
  - Builds release notes in both the original (English) version and translated versions (if any).

Note that the scripts in the tasks use zanata-cli to pull and push translation content.

Debugging job scripts

See *Debugging job scripts*.

1.14 Debugging job scripts

While it would be rare, you may would like to debug the translation job scripts. This section describes tips for such cases. `propose_translation_update.sh` is mainly covered.

> **Note**
>
> It is not a complete document for debugging. It was just written as a note. Feel free to add more topics.

Environment

It is better to use the same environment (e.g., Linux distribution version) used in the translation job. To find the information, check `nodeset` of the `propose-translation-update` job definition in zuul configuration in openstack/project-config repository. As of Feb 2021, `ubuntu-bionic` is used.

Preparations

You can prepare dependencies by following prepare-zanata-client role.

What you need are:

- Installing Zanata CLI and its dependencies. Ensure `zanata-cli` command is found in your path.
- Prepare `~/.config/zanata.ini`. See *Zanata CLI* for more detail.
- Copy the translation scripts to your working directory from openstack-zuul-jobs/roles/prepare-zanata-client/files. In addition, you need to copy `common.sh` from project-config/roles/copy-proposal-common-scripts/files to the same working directory.

Commenting out CI-specific codes

The job scripts contain codes specific to OpenStack CI. For example, there is a code to communicate with OpenStack gerrit and the account is hardcoded. It looks convenient to comment out such code blocks to debug the scripts in a local env. It should be useful unless you are debuging a CI-specific issue.

Code blocks which look better to be commented out are:

- common_translation_update.sh
  - `trap "finish" EXIT` (It depends on testrepository and usually fails in local envs)
- propose_translation_update.sh

- `setup_review` and a logic to check an existing change (It is unnecessary as we do not propose a real change to gerrit. The account used is hardcoded and it always fails in local runs.)
- `send_patch` (It is unnecessary as we do not want to propose a real change to gerrit.)
- (optional) The main logic (from `case "$PROJECT" in` to `filter_commits`) (we will debug the main logic, so perhaps we would like to run the script piece by piece.)

The diff would be like http://paste.openstack.org/show/802260/.

Copying upper-constraints.txt

The job scripts assume that upper-constraints.txt exists in the top directory of the target project repository.

```
$ cd $PROJECT_DIR
$ wget https://releases.openstack.org/constraints/upper/master
```

Note that you need to adjust the URL of the upper-constraints file when you work on a stable branch.

Creating ~/.venv virtualenv

The job scripts assume that required python modules are installed in `~/.venv` virtualenv. In the zuul job, this virtualenv is prepared via `ensure-sphinx` and `ensure-babel` roles in zuul/zuul-jobs. You need to do the same.

Note that doc/requirements.txt in most projects are almost samer so perhaps you can reuse this virtualenv for most projects. If it does not work, consider recreating the virtualenv.

```
$ python3 -m venv ~/.venv
$ . ~/.venv/bin/activate
# $PROJECT_DIR is a target project repository like horizon, ironic-ui ....
(.venv) $ cd $PROJECT_DIR
# Install sphinx related modules ensure-sphinx installs
(.venv) $ pip install -r doc/requirements.txt -c ../requirements/upper-
↪constraints.txt
# Install modules ensure-babel installs
(.venv) $ pip install Babel lxml pbr requests -c ../requirements/upper-
↪constraints.txt
(.venv) $ deactivate
```

Loading scripts

```
# It is better to start a new shell when debugging the script
$ bash
> cd $PROJECT_REPO
# The arguments are: {{ zuul.project.short_name }} {{ zuul.branch }} {{ zuul.
↪job }} $HORIZON_REPO
> . $WORKDIR/propose_translation_update.sh ironic-ui master propose-
↪translation-update ../horizon
>
```

Simulating translation changes

If you would like to simulate translation changes, you can download translations in advance (using zanata-cli) and modify them as you want. In such case, you can modify the script as follows:

```
--- a/roles/prepare-zanata-client/files/common_translation_update.sh
+++ b/roles/prepare-zanata-client/files/common_translation_update.sh
@@ -734,7 +732,8 @@ function pull_from_zanata {
     # Since Zanata does not currently have an option to not download new
     # files, we download everything, and then remove new files that are not
     # translated enough.
-    zanata-cli -B -e pull
+    #zanata-cli -B -e pull
+    cp -r $DOWNLOAD_TRANSLATIONS/$project/* .

     # We skip directories starting with '.' because they never contain
     # translations for the project (in particular, '.tox'). Likewise
```

You can download translations as below. `zanata.xml` will be created once you run the propose_translation_update.sh (or you can prepare it by following *Zanata CLI* Project configuration).

```
$ cd $DOWNLOAD_TRANSLATIONS/$project
$ zanata-cli -B -e pull --project-config $PROJECT_DIR/zanata.xml
```

1.15 Project Repository Setup to Enable Translation Infrastructure

Note

This document was moved from OpenStack Project Creators Guide who want to enable translation infrastructure for their project. Before proceeding, target translation project with corresponding versions (master and stable branches) should be created on Zanata and I18n team will help with this.

Once you have your project repository set up, you might want to enable translations. For this, you first need to mark all strings so that they can be localized, for Python projects use oslo.i18n for this and follow the guidelines.

Note that this is just enabling translations, the actual translations are done by the i18n team, and they have to prioritize which projects to translate.

First enable translation in your project, depending on whether it is a Django project, a Python project or a ReactJS project.

Note

The infra scripts consider a project as a Django project when your repository name ends with `-dashboard`, `-ui`, `horizon` or `django_openstack_auth`. Otherwise your project will be recognized as a Python project. Projects using ReactJS need special treatment.

If your repository structure is more complex, for example, with multiple python modules, or with both Django and Python projects, see *More complex cases* as well.

Python Projects

For translation of strings in Python files, only a few changes are needed inside a project.

> **Note**
>
> Previously `setup.cfg` needed sections `compile_catalog`, `update_catalog`, and `extract_messages` and a `babel.cfg` file. These are not needed anymore and can be removed.

Update your `setup.cfg` file. It should contain a `packages` entry in the `files` section:

```
[files]
packages = ${MODULENAME}
```

Replace `${MODULENAME}` with the name of your main module like `nova` or `novaclient`. Your i18n setup file, normally named `_i18n.py`, should use the name of your module as domain name:

```
_translators = oslo_i18n.TranslatorFactory(domain='${MODULENAME}')
```

Django Projects

Update your `setup.cfg` file. It should contain a `packages` entry in the `files` section:

```
[files]
packages = ${MODULENAME}
```

Create file `babel-django.cfg` with the following content:

```
[python: **.py]
[django: **/templates/**.html]
[django: **/templates/**.csv]
```

Create file `babel-djangojs.cfg` with the following content:

```
[javascript: **.js]
[angular: **/static/**.html]
```

ReactJS Projects

Three new dependencies are required : `react-intl`, `babel-plugin-react-intl`, and `react-intl-po`.

Update your `package.json` file. It should contain references to the `json2pot` and `po2json` commands.

```
"scripts": {
    ...
    "json2pot": "rip json2pot ./i18n/extracted-messages/**/*.json -o ./i18n/
↪messages.pot",
    "po2json": "rip po2json -m ./i18n/extracted-messages/**/*.json"
    }
```

The translated PO files will converted into JSON and placed into the `./i18n/locales` directory.

You need to update the infra scripts as well to mark a repository as ReactJS project for translation, for details see *Translation infrastructure tasks and scripts*.

Add Translation Server Support

Propose a change to the `openstack/project-config` repository including the following changes:

1. Set up the project on the translation server.

   Edit file `gerrit/projects.yaml` and add the `translate` option:

   ```
   - project: openstack/<projectname>
     description: Latest and greatest cloud stuff.
     options:
       - translate
   ```

2. Add the jobs to your pipelines.

   Edit file `zuul.d/projects.yaml` and add a template which defines translation jobs to your repository:

   ```
   - project:
       name: openstack/<projectname>
       templates:
         - translation-jobs-master-stable
   ```

   The translation team is translating stable branches only for GUI projects, so for horizon and its plugins.

   If the repository is a GUI project, use the `translation-jobs-master-stable` template. Otherwise use the `translation-jobs-master-only` template.

When submitting the change to `openstack/project-config` for review, use the `translation_setup` topic so it receives the appropriate attention:

```
$ git review -t translation_setup
```

With these changes merged, the strings marked for translation are sent to the translation server after each merge to your project. Also, a periodic job is set up that checks daily whether there are translated strings and proposes them to your project together with translation source files. Note that the daily job will only propose translated files where the majority of the strings are translated.

Checking Translation Imports

As a minimal check that the translation files that are imported are valid, you can add to your lint target (`pep8` or `linters`) a simple `msgfmt` test:

```
$ bash -c "find ${MODULENAME} -type f -regex '.*\.pot?' -print0| \
        xargs -0 -n 1 --no-run-if-empty msgfmt --check-format -o /dev/null"
```

Note that the infra scripts run the same test, so adding it to your project is optional.

More complex cases

The infra scripts for translation setup work as follows:

- The infra scripts recognize a project type based on its repository name. If the repository name ends with `-dashboard`, `-ui`, `horizon` or `django_openstack_auth`, it is treated as a Django project. Otherwise it is treated as a Python project.
- If your repository declares multiple python modules in `packages` entry in `[files]` section in `setup.cfg`, the infra scripts run translation jobs for each python module.

We strongly recommend to follow the above guideline, but in some cases this behavior does not satisfy your project structure. For example,

- Your repository contains both Django and Python code.
- Your repository defines multiple python modules, but you just want to run the translation jobs for specific module(s).

In such cases you can declare how each python module should be handled manually in `setup.cfg`. Python modules declared in `django_modules` and `python_modules` are treated as Django project and Python project respectively. If `django_modules` or `python_modules` entry does not exist, it is interpreted that there are no such modules.

```
[openstack_translations]
django_modules = module1
python_modules = module2 module3
```

You also need to setup your repository following the instruction for Python and/or Django project above appropriately.

Project Documentation Translation

Documents on the project following with OpenStack documentation structure can be translated into multiple languages. List of project repositories for documentation translation is maintained in the `DOC_TARGETS` variable within common_translation_update.sh translation infrastructure script file.

1.16 Reviewing translation imports

Note

This document explains to reviewers details about the automatic import of translations from Zanata.

If you are unfamiliar with translations in OpenStack, read the Project Team Guide on Internationalization and Translation first.

This document gives additional information.

How are translations handled?

Translators translate repositories using the translation server which runs the Zanata software.

Every day, new translations get imported into the repositories using a proposal job. These need to have a review on whether the bot worked properly. You can see all open reviews in Gerrit. The subject of these patches is always Imported Translations from Zanata.

Reviewing

If you are reviewing these translations, keep the following in mind:

- The change is done by a bot. If anything looks wrong with it, you need to actively reach out to the *OpenStack I18n team* and point the problem out.
- The goal of the review is that the structure of the change is fine, its not that the strings are translated properly. The review of translated strings is done by teams using the translation server.
- If you notice bad translations in a language, file a bug (see *Reporting translation bugs*). Then the translation team will update the translation. We recommend to still import this change as is and import later the fix to not block other valid translations to merge.
- Nobody should change translation files (the *.po* files in the *locale* directory) besides the bot. The next automatic import will override any change again. Therefore, leave a `-1` vote on any such changes and point developer to this document.
- The proposal bot also removes files, it removes files that have very few translations in them. Note that no translations will be lost, they are still in the translation server.

  Also, release notes translations are only needed on master since release notes are published only from master, the translations get removed on stable branches.
- Most teams have single approval for translation imports instead of two core reviewers.

1.17 Team activities with release

This page documents what I18n team concerns and which things are needed to do with OpenStack releases. Each OpenStack release has around 6-month cycle and the corresponding schedule is available on https://releases.openstack.org/ (e.g., https://releases.openstack.org/ocata/schedule.html describes Ocata release schedule).

One of main goals in I18n team is to incorporate translated strings into a new release so that more global users experience translated version of OpenStack. To accomplish this goal, some of team activities need to be aligned with a release schedule.

Note

I18n team sets target projects to be translated and prioritizes during around the PTG. Current translation plan and priority are available on translation dashboard.

Note

The terms in this page follow release schedule pages. You can see how OpenStack project teams product artifacts with release management on Release Management document.

Projects affecting StringFreezes

Horizon and Horizon plugins are the main targets which affect StringFreezes. Note that StringFreezes are applied to I18n team target projects and with `cycle-with-rc` or `cycle-with-intermediary` release model.

1. [Project] Release milestone-3. `Soft StringFreeze` is in effect.

2. [Translator] Start translations for the release.
    - Translate **master** version on Zanata.
3. [I18n SIG Chair] Call for translation
4. [I18n SIG Chair] Coordinate release and translation import schedule of individual projects with SIG Chair or I18n liaison.
5. [Project] Release RC1 and create a stable branch. `Hard StringFreeze` is in effect.
6. [Translator] It is suggested to complete translation work by Monday or Tuesday of the Final RC week.
    - It is encouraged to share translation progress with I18n team and/or corresponding project team(s) to make sure shipping the translation.
7. [Project] RC2 or RC3 release will be shipped with latest translations. Final RC release will happen one week before the official release week.
8. [Project] Official release!
9. [Zanata admin] Create a stable version after release.
10. [Infra] Setup translation jobs such as `translation-jobs-newton` to import translations for stable branches.
    - This should be done after a stable version on Zanata is created.
11. After the official release, translating master version is welcome for upstream translation contribution, but the original strings may be changed frequently due to upstream development on the projects.
12. On the other hand, translating stable version as upstream contribution is not encouraged after the translated strings are packaged with releases. The stable version will be closed earlier than or around EOL.
13. If there is a translation bug on a stable version in Zanata, it is highly recommended to fix the same translation bug on the corresponding string in the master version.

Documentation projects

All upstream OpenStack documents in `openstack-manuals` and project repositories use `master` branch and can have stable branches depending on release models and maintaining documents in different project teams. Cross-project liaisons are strongly encouraged to communicate with I18n team so that I18n team can discuss and decide which stable documents are the target in I18n team and when translators start to translate stable documents.

1. Translations on master version in Zanata are normally recommended for upstream contribution.
2. After the documents are translated and reviewed well, updating existing `www/<lang>/index.html` or creating a new language landing page is needed.

Other projects supported with translation jobs

I18n team helps release activities on other projects if the projects have cross-project liaison on I18n team and upon request.

1.18 About this guide

This page describes the conventions and tips on writing this guide itself.

Convention

The guide is written in reStructuredText (RST) markup syntax with Sphinx extensions. Most conventions follow those of the openstack-manuals project.

The followings are useful links when writing documents in RST.

- Sphinx documentation
- Quick reStructuredText

Titles

The convention for heading levels is as follows:

```
=========
Heading 1
=========

Overline and underline with equal signs for heading 1 sections.
This level is reserved for the title in a document.

Heading 2
---------

Underline with dashes for heading 2 sections.

Heading 3
~~~~~~~~~~

Underline with tildes for heading 3 sections.
```

Translation

This guide itself is I18n-ed and you can translate it into your language. To translate it, visit i18n project in Zanata. Document **doc** in **i18n** project corresponds to this guide. You can translate it in the same way as you do for other projects like dashboard or manuals. Once the translation progress becomes higher than the threshold (For more information on the threshold, see *Translation jobs*), the translated version of the guide will be published.

2 Getting in touch

2.1 Language translation team

Each language team has useful information to help our translation efforts. It is worth visited. You can find your language team at https://wiki.openstack.org/wiki/I18nTeam/team.

2.2 OpenStack I18n team

- Mailing List: openstack-i18n@lists.openstack.org
- IRC channel: `#openstack-i18n` on OFTC