QGIS Documentation Guidelines

QGIS Project

Oct 24, 2025

CONTENTS

QGIS Documentation is available at https://www.qgis.org/resources/hub/#documentation. As the writing process is going on, a build is automatically run every day (see bottom of the page for exact time) for all supported versions (testing, Long Term Release (LTR) and next-to-be LTR).

QGIS Documentation source files are available at https://github.com/qgis/QGIS-Documentation. They are mainly written using the reStructuredText (reST) format syntax, coupled with some scripts from the Sphinx toolset to post-process the HTML output.

The following chapters will guide you through learning:

- how to manage the documentation source files using git system and the GitHub platform on which they are stored
- how to modify the texts, provide screenshots… in a compliant way
- how to share and ensure your modifications are pushed to the official docs.

If you are looking for general information about how to contribute to the QGIS project, you may find help at Get Involved in the QGIS Community.

CHAPTER

ONE

A STEP BY STEP CONTRIBUTION

Note

Though QGIS-Documentation is used to demonstrate the process, all commands and steps shown below also apply to QGIS-Website.

If you are reading these lines, it is certainly because you are willing to contribute to writing QGIS documentation and are looking for a how-to. You have come to the right place! The current document will guide you through the different ways to achieve this objective, showing you the main steps to follow, the tricks you can use and the traps you should be aware of.

For any help, do not hesitate to either ask in a comment on the issue report you are trying to fix or write to the QGIS-community-team list. Read general information on QGIS community support.

Let's now dive into the process.

Documentation sources are stored using the Git version control system and are available on GitHub at https://github.com/qgis/QGIS-Documentation. A list of issues to fix and features to explain can be found at https://github.com/qgis/QGIS-Documentation/issues.

Tip

If you are a first-time contributor and do not know where to start from, you may be interested in tackling our welcoming issue reports.

There are two main ways, not mutually exclusive, to modify the files:

1. *Using the GitHub web interface*
2. *Using Git command line tools*.

1.1 Using the GitHub web interface

The GitHub web interface allows you to do the following:

- edit files
- preview and commit your changes
- make a pull request to have your changes inserted into the main repository
- create, update, or delete branches

If you are not yet familiar with Git and GitHub vocabulary, you may want to read the GitHub Hello-world project to learn some basic vocabulary and actions that will be used below.

Note

If you are fixing a reported issue

If you are making changes to fix an issue, add a comment to the issue report to assign it to yourself. This will prevent more than one person from working on the same issue.

1.1.1 Fork QGIS-Documentation

Assuming you already have a GitHub account, you first need to fork the source files of the documentation.

Navigate to the QGIS-Documentation repository page and click on the 

button in the upper right corner.

In your GitHub account you will find a QGIS-Documentation repository (`https://github.com/<YourName>/QGIS-Documentation`). This repository is a copy of the official QGIS-Documentation repository where you have full write access and you can make changes without affecting the official documentation.

1.1.2 Make changes

There are different ways to contribute to QGIS documentation. We show them separately below, but you can switch from one process to the other without any harm.

Alternative 1: Use the `Edit on GitHub` shortcut

Pages on the QGIS documentation website can be edited quickly and easily by clicking on the `Edit on GitHub` link:

- at the top right of each page,
- or in the drop-down menu at the bottom of the left sidebar.

1. This will open the file in the `qgis:master` branch with a message at the top of the page telling you that you don't have write access to this repo and your changes will be applied to a new branch of your repository.

   Note

   If you do have write access to the QGIS repository, no message is displayed and your changes will likely be saved directly in the main repository instead of your fork. However, to keep the repository clean, it is recommended that even contributors with write access work from their forks when editing documentation.

2. Do your changes. Since the documentation is written using the reStructureText syntax, depending on your changes, you may need to rely on the *writing guidelines*.
3. Once you have finished modifying the file, press *Commit changes…* and add a short title explaining your changes. You can also provide a more detailed explanation, if necessary.
4. Press *Propose changes*, saving the modifications to your repository. This will generate a dedicated branch (`patch-xxx`) in your repository and the web browser will open the *Comparing changes* page.
   - If the issue you are addressing does not need any further changes, jump to the *Share your changes via Pull Request* section below.
   - If there are additional changes that you want to make before submitting them to QGIS, follow these steps:
     1. Navigate to your fork of QGIS-Documentation (`https://github.com/<YourName>/QGIS-Documentation`)
     2. Click on 

and search for the new `patch-xxx` branch.
     3. Select it. The 

button will now display *Branch: patch-xxx*.
     4. Jump down to the *Modify files* section below.

Alternative 2: Create an ad hoc branch in your documentation repository

You can edit files directly from your fork of the QGIS Documentation.

First, make sure that your `master` branch is up to date with upstream, i.e. `qgis/QGIS-Documentation:master` branch. To do so:

1. Go to the main page of your repository, i.e. `https://github.com/<YourName>/QGIS-Documentation`. The `master` branch should be active with a mention whether it is up to date with `qgis/QGIS-Documentation:master` or not.

   If it has commits ahead the upstream branch, you better use the previous *shortcut button alternative* until you align your `master` branch.

   If it only has commits behind:

1. Expand the *Sync fork* drop-down menu on the right. You can:
   - *Compare* the branches and see new changes in the upstream branch you do not yet have in yours
   - *Fetch and merge*: brings changes from the upstream branch into yours.
2. Click *Fetch and merge*: after the process, your branch is mentioned as up to date with `qgis/QGIS-Documentation:master`.

2. Click on Branch: master in the upper left corner of your forked QGIS-Documentation repository and enter a unique name in the text field to create a new branch. It is advised that the name of the new branch relates to the problem you intend to fix. Convenient for identifying the branch later.
3. Press *Create branch <branch_name> from master*. The Branch: master button should now display *branch_name*.
4. You are ready to start new changes on top of your new branch.

Attention

Do your changes in an ad hoc branch, never in the `master` branch

By convention, avoid making changes in your `master` branch except when you merge the modifications from the `master` branch of `qgis/QGIS-Documentation` into your copy of the QGIS-Documentation repository. Separate branches allow you to work on multiple problems at the same time without interfering with other branches. If you make a mistake you can always delete a branch and start over by creating a new one from the master branch.

1.1.3 Modify files

1. Make sure the branch you want to modify is active: its name is displayed in the top left menu or in the page URL.
2. Browse the source files to the file that needs modifications.
3. Press the Edit this file button.
4. Update the text following the *writing guidelines*
5. Once you have finished modifying the file, press *Commit changes…* and add a short title explaining your changes. You can also provide a more detailed explanation, if necessary.
6. Make sure *Commit directly to the <branch_name> branch* is selected.
7. Press *Commit changes* to save your modifications in the branch.
8. To update or add a new image file:
   1. Browse your branch to the folder of the file to update or add.
   2. Top right, use the *Add files* ► *Upload files* menu to select and load files from your drive. You can find hints for formatting an image for the docs at *Managing Screenshots*.
   3. If a file of the same name already exists, it gets overwritten.
   4. Commit once again your changes to the target branch.
9. Repeat the previous steps for any other files that need to be updated to fix the issue.

1.1.4 Share your changes via Pull Request

You need to make a pull request to integrate your changes into the official documentation.

Start a new pull request

1. If you used the `Edit on GitHub` link, after you commit your changes, GitHub automatically opens a new page comparing the changes you made in your `patch-xxx` branch to the `qgis/QGIS-Documentation:master` branch.
2. In all the cases, the comparing changes page can be reached by going to the "Pull requests" page of your repository and click on *New pull request*.

Compare changes

If you see two dialog boxes, one that says `base:master` and the other `compare:branch_name` (see figure), this will only merge your changes from one of your branches to your master branch. To fix this click on the *compare across forks* link.

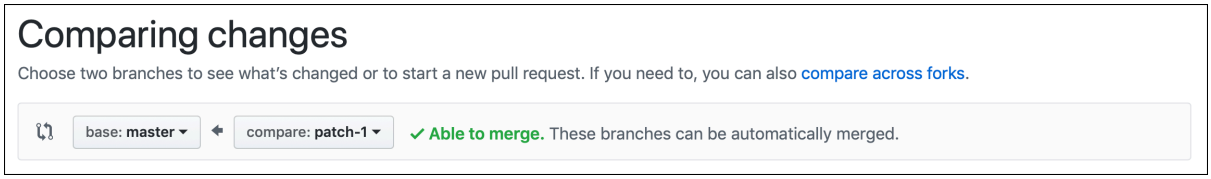


Fig. 1.1: If your *Comparing changes* page looks like this, click on the *compare across forks* link.

You should see four drop-down menus. These will allow you to compare the changes that you have made in your branch with the master branch that you want to merge into. They are:

- **base fork**: the fork that you want to merge your changes into
- **base**: the branch of the base fork that you want to merge your changes into
- **head fork**: the fork that has changes that you want to incorporate into the base fork
- **compare**: the branch with those changes

Select `qgis/QGIS-Documentation` as the base fork with `master` as base, set the head fork to your repository `<YourName>/QGIS-Documentation`, and set compare to your modified branch.

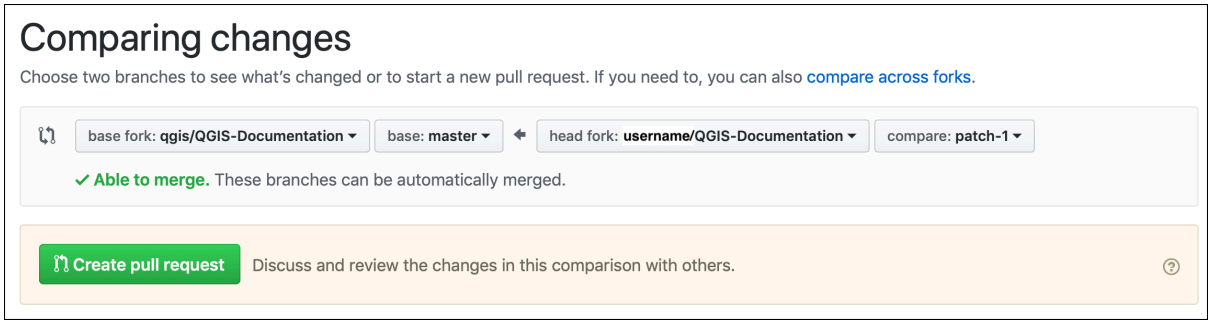


Fig. 1.2: Comparing changes between `qgis/QGIS-Documentation` and your repository

A green check with the words **Able to merge** shows that your changes can be merged into the official documentation without conflicts.

Click the *Create pull request* button.

Warning

If you see ✕ Can't automatically merge.

This means that there are conflicts. The files that you are modifying are not up to date with the branch you are targeting because someone else has made a commit that conflicts with your changes. You can still create the pull request but you'll need to fix any *conflicts* to complete the merge.

Tip

Though being translated, the latest version of QGIS documentation is still maintained and existing issues are fixed. If you are fixing issues for a different release, change **base** from `master` to the appropriate `release_...` branch in the steps above.

Describe your pull request

A text box will open: fill in a title and any relevant description of the changes you are proposing.

If this relates to a particular issue, add the issue number to your comments. This is done by entering # and the issue number (*e.g.* `#1234`). If preceded by terms like `fix` or `close`, the concerned issue will be closed as soon as the pull request is merged.

Add links to any documentation pages that you are changing.

Click on *Create pull request*.

Review and comment pull request

As seen above, anyone can submit modifications to the documentation through pull requests. Likewise anyone can review pull requests with questions and comments. Perhaps the writing style doesn't match the project guidelines, the change is missing some major details or screenshots, or maybe everything looks great and is in order. Reviewing helps to improve the quality of the contribution, both in form and substance.

To review a pull request:

1. Navigate to the pull requests page and click on the pull request that you want to comment on.
2. At the bottom of the page you will find a text box where you can leave general comments about the pull request.
3. To add comments about specific lines,
    1. Click on Files changed (Display the source diff) and find the file you want to comment on. You may have to click on < > to see the changes.
    2. Scroll to the line you want to comment on and click on the  . That will open a text box allowing you to leave a comment.

Specific line comments can be published either:

- as single comments, using the *Add single comment* button. They are published as you go. Use this only if you have few comments to add or when replying to another comment.
- or as part of a review, pressing the *Start a review* button. Your comments are not automatically sent after validation, allowing you to edit or cancel them afterwards, to add a summary of the main points of the review or global instructions regarding the pull request and whether you approve it or not. This is the convenient way since it's more flexible and allows you to structure your review, edit the comments, publish when you are ready and send a single notification to the repository followers and not one notification for each comment. Get more details.

Fig. 1.3: Commenting a line with a change suggestion

Line comments can embed suggestions that the pull request writer can apply to the pull request. To add a suggestion, click the Insert a suggestion button on top of the comment text box and modify the text within the suggestion block.

Tip

Prefer committing suggestions to your pull request as a batch

As a pull request author, when directly incorporating reviewers' feedback in your pull request, avoid using the *Commit suggestion* button at the bottom of the comment when you have many suggestions to address and prefer adding them as a batch commit, that is:

1. Switch to the Files changed tab
2. Press *Add suggestion to batch* for each rewording you'd like to include. You will see a counter increasing as you go.
3. Press any of the *Commit suggestions* button when you are ready to apply the suggestions to your pull request, and enter a message describing the changes.

This will add all the modifications to your branch as a single commit, resulting in a more legible history of changes and less notifications for the repository followers. Incidentally, proceeding as this will also save you many clicks.

Make corrections

A new pull request will automatically be added to the Pull requests list. Other editors and administrators will review your pull request and they may make suggestions or ask for corrections.

A pull request will also trigger automated build checks (eg, for rst formatting, python code syntaxes), and reports are displayed at the bottom of the page. If an error is found, a red cross will appear next to your commit. Click on the red cross or on `Details` in the summary section at the bottom of the pull request page to see the details of the error. You'll have to fix any reported errors or warnings before your changes are committed to the `qgis/QGIS-Documentation` repository.

You can make modifications to your pull request until it is merged with the main repository, either to improve your request, to address requested modifications, or to fix a build error:

1. Click on the Files changed tab in your pull request page
2. Press … menu on the right of the filename that you want to modify, and select *Edit file*.

3. The file opens in edit mode; add your modifications and commit.

Any additional changes will be automatically added to your pull request if you make those changes to the same branch that you submitted in your pull request. For this reason, you should only make additional changes if those changes relate to the issue that you intend to fix with that pull request.

If you want to fix another issue, create a new branch for those changes and repeat the steps above.

An administrator will merge your contribution after any build errors are corrected, and after you and the administrators are satisfied with your changes.

1.1.5 Delete your merged branch

You can delete the branch after your changes have been merged. Deleting old branches saves you from having unused and outdated branches in your repository.

1. Navigate to your fork of the QGIS-Documentation repository (`https://github.com/<YourName>/QGIS-Documentation`).
2. Click on the *Branches* tab. Below *Your branches* you'll see a list of your branches.
3. Click on the  Delete this branch icon to delete any unwanted branches.

1.2 Using Git command line tools

The GitHub web interface is an easy way to update the QGIS-documentation repo with your contributions, but it doesn't offer tools to:

- group your commits and clean your change history
- fix possible conflicts with the main repo
- build the documentation to test your changes

You need to install Git on your hard drive in order to get access to more advanced and powerful tools and have a local copy of the repository. Some basics you may often need are exposed below. You'll also find rules to care about even if you opt for the web interface.

In the code samples below, lines beginning with `$` show commands you should type while `#` are comments.

1.2.1 Local repository

Now you are ready to get a local clone of **your** QGIS-Documentation repository.

You can clone your QGIS repository using the web URL as follows:

```
# move to the folder in which you intend to store the local repository
$ cd ~/Documents/Development/QGIS/
$ git clone https://github.com/<YourName>/QGIS-Documentation.git
```

The former command line is simply an example. You should adapt both the path and the repository URL, replacing `<YourName>` with your GitHub user name.

Check the following:

```
# Enter the local repository
$ cd ./QGIS-Documentation
$ git remote -v
origin  https://github.com/<YourName>/QGIS-Documentation.git (fetch)
origin  https://github.com/<YourName>/QGIS-Documentation.git (push)
```

(continues on next page)

(continued from previous page)

```
$ git branch
* master
```

- *origin* is the name of the remote repository of your QGIS-Documentation repository.
- *master* is the default main branch. You should never use it to contribute! **Never!**

Alternatively you can clone your QGIS repository using the SSH protocol:

```
# move to the folder in which you intend to store the local repository
$ cd ~/Documents/Development/QGIS/
$ git clone git@github.com:<YourName>/QGIS-Documentation.git
```

> **Tip**
>
> **Permission denied (publickey) error?**
>
> If you get a Permission denied (publickey) error with the former command, there may be a problem with your SSH key. See GitHub help for details.

Check the following if you used the SSH protocol:

```
# Enter the local repository
$ cd ./QGIS-Documentation
$ git remote -v
origin  git@github.com:<YourName>/QGIS-Documentation.git (fetch)
origin  git@github.com:<YourName>/QGIS-Documentation.git (push)
$ git branch
* master
```

You can start to work here but in the long term process you will get a lot of issues when you will push your contribution (called Pull Request in GitHub process) as the master branch of the qgis/QGIS-Documentation repository will diverge from your local/remote repository. You then need to keep track of the main remote repository and work with branches.

1.2.2 Add another remote repository

To be able to follow the work in the main project, add a new remote repository in your local repository. This new remote repository is the QGIS-Documentation repository from QGIS project:

```
$ git remote add upstream https://github.com/qgis/QGIS-Documentation.git
$ git remote -v
origin  https://github.com/<YourName>/QGIS-Documentation.git (fetch)
origin  https://github.com/<YourName>/QGIS-Documentation.git (push)
upstream        https://github.com/qgis/QGIS-Documentation.git (fetch)
upstream        https://github.com/qgis/QGIS-Documentation.git (push)
```

Similarly, you can use the SSH protocol to add a remote repository in your local repository:

```
$ git remote add upstream git@github.com:qgis/QGIS-Documentation.git
$ git remote -v
origin  git@github.com:<YourName>/QGIS-Documentation.git (fetch)
origin  git@github.com:<YourName>/QGIS-Documentation.git (push)
upstream        git@github.com:qgis/QGIS-Documentation.git (fetch)
upstream        git@github.com:qgis/QGIS-Documentation.git (push)
```

So now you have the choice between two remote repository:

- *origin* to push your local branch in **your** remote repository

- *upstream* to merge (if you have right to do so) your contribution to the official one OR to update your master branch of local repository from the master branch of the official repository.

> **Note**
>
> *upstream* is just a label, a kind of standard name but you can call it as you want.

1.2.3 Update your base branch

Before working on a new contribution, you should always update your master branch in your local repository. Assuming you are willing to push changes to the testing documentation, run the following command lines:

```
# switch to master branch (it is easy to forget this step!)
$ git checkout master
# get "information" from the master branch in the upstream repository
# (aka qgis/QGIS-Documentation's repository)
$ git fetch upstream master
# merge update from upstream/master to the current local branch
# (which should be master, see step 1)
$ git merge upstream/master
# update **your** remote repository (aka <YourName>/QGIS-Documentation)
$ git push origin master
```

Now you have your local and remote repositories which both have their `master` branch up to date with the official `master` branch of QGIS-Documentation. You can start to work on your contribution.

> **Note**
>
> Switch the branch if you wish to contribute to released doc
>
> Along with the testing documentation, we continue to fix issues in the latest release, meaning that you can also contribute to it. Follow the previous section sample code, replacing `master` with the corresponding branch of the latest documentation.

1.2.4 Contribute to your production branch

Now that your base branch is updated, you need to create a dedicated branch in which you add your contribution. Always work on a branch other than the base branch! Always!

```
# Create a new branch
$ git checkout -b myNewBranch
# checkout means go to the branch
# and -b flag creates a new branch if needed, based on current branch
# Let's check the list of existing branches (* indicates the current branch)
$ git branch
master
release_2.18
...
* myNewBranch
# You can now add your contribution, by editing the concerned file(s)
# with any application (in this case, vim is used)
$ vim myFile
# once done
$ git add myFile
$ git commit
```

Few words about commit/push commands:

- try to commit only one contribution (atomic change) i.e. address only one issue
- try to explain carefully what you change in the title of your commit and in the description. The first line is a title and should start by an upper case letter and have 80 characters length, don't end with a `.`. Be concise. Your description can be longer, end with a `.` and you can give much more details.
- use a `#` with a number to refer to an issue. Prefix with `Fix` if you fix the ticket: your commit will close the ticket.

Now that your changes are saved and committed in your local branch, you need to send them to your remote repository in order to create pull request:

```
$ git push origin myNewBranch
```

1.2.5 Share your Changes

Now you can go to your GitHub repository and *create a Pull Request* as exposed in a previous section. Ensure you create a PR from your branch to the remote branch you are targetting in the official QGIS-Documentation repository.

1.2.6 Clean-up your local and remote repository

After your PR has been merged into the official QGIS-Documentation, you can delete your branch. If you work a lot this way, in few weeks you will get a lot of unuseful branches. So keep your repository clean this way:

```
# delete local branch
$ git branch -d myNewBranch
# Remove your remote myNewBranch by pushing nothing to it
$ git push origin :myNewBranch
```

And do not forget to update the `master` branch in your local repository!

1.3 Further reading

- Other than the GitHub web interface and the Git command line tools exposed above, there are also GUI applications you can use to create and manage your contributions to the documentation.
- When the changes in the pull request are conflicting with recent changes pushed to the target branch, the conflicts need to be resolved before a merge is possible:
  - if the conflict relates to few competing lines, a *Resolve conflicts* button is available in the GitHub pull request page. Press the button and resolve the issue as explained at Resolving a merge conflict on GitHub
  - if the conflict involves files renaming or removal, then you'd need to resolve the conflict using Git command lines. Typically, you have to first rebase your branch over the target branch using `git rebase targetBranch` call and fix the conflicts that are reported. Read more at Resolving a merge conflict using the command line
- Sometimes, at the end of the proofreading process, you may end up with changes split into multiple commits that are not necessarily worth it. Git command lines help you squash these commits to a smaller number and more meaningful commit messages. Some details at Using git rebase on the command line

CHAPTER

TWO

WRITING GUIDELINES

In general, when creating reST documentation for the QGIS project, please follow the Python documentation style guidelines. For convenience, we provide a set of general rules we rely on for writing QGIS documentation below.

2.1 Writing Documentation

2.1.1 Headlines

To each webpage of the documentation corresponds a `.rst` file.

Sections used to structure the text are identified through their title which is underlined (and overlined for the first level). Same level titles must use same character for underline adornment. In QGIS Documentation, you should use following styles for chapter, section, subsection and minisec.

```
********
Chapter
********

Section
=======

Subsection
----------

Minisec
.......

Subminisec
^^^^^^^^^^
```

2.1.2 Lists

Lists are useful for structuring the text. Here are some simple rules common to all lists:

- Start all list items with a capital letter
- Do not use punctuation after list items that only contain a single simple sentence
- Use period (`.`) as punctuation for list items that consist of several sentences or a single compound sentence

2.1.3 Indentation

Indentation in reStructuredText should be aligned with the list or markup *marker*. It is also possible to create block quotes with indentation. See the Specification

```
#. In a numbered list, there should be
   three spaces when you break lines
#. And next items directly follow

   * Nested lists
   * Are also possible
   * And when they also have
     a line that is too long,
     the text should be naturally
     aligned
   * and be in their own paragraph

However, if there is an unindented paragraph, this will reset the numbering:

#. This item starts at 1 again
```

2.1.4 Inline Tags

You can use tags to emphasize items.

- **Menu GUI**: to mark a complete sequence of menu selections, including selecting submenus and choosing a specific operation, or any subsequence of such a sequence.

```
:menuselection:`menu --> submenu`
```

- **Dialogs and Tab titles**: Labels presented as part of an interactive user interface including window titles, tab titles, button and option labels.

```
:guilabel:`title`
```

- **Filenames and directories**

```
:file:`README.rst`
```

- **Icons with popup text**

```
|icon| :sup:`popup_text`
```

(see *image* below).

- **Keyboard shortcuts**

```
:kbd:`Ctrl+B`
```

will show `Ctrl+B`

When describing keyboard shortcuts, the following conventions should be used:

  - Letter keys are displayed using uppercase: `S`
  - Special keys are displayed with an uppercase first letter: `Esc`
  - Key combinations are displayed with a + sign between keys, without spaces: `Shift+R`

- **User text**

```
``label``
```

- **Layer names**

When referring to layers, format as inline code:

```
``layer name``
```

2.1.5 Labels/references

Anchors inside the text can be used to create hyperlinks to sections or pages.

The example below creates the anchor of a section (e.g., Label/reference title)

```
.. _my_anchor:

Label/reference
---------------
```

To call the reference in the **same page**, use

```
see my_anchor_ for more information.
```

which will return:

see *my_anchor* for more information.

Notice that it will jump to the line/thing following the 'anchor'. You do not need to use apostrophes, but you do need to have empty lines after the anchor.

Another way to jump to the same place **from anywhere in the documentation** is to use the `:ref:` role.

```
see :ref:`my_anchor` for more information.
```

which will create a link with the caption instead (in this case the title of this section!):

see *Labels/references* for more information.

So, reference 1 (*my_anchor*) and reference 2 (*Labels/references*). Because the reference often displays a full caption, it is not really necessary to use the word *section*. Note that you can also use a custom caption to describe the reference:

```
see :ref:`Label and reference <my_anchor>` for more information.
```

which returns:

see *Label and reference* for more information.

2.1.6 Figures and Images

Pictures

To insert an image, use

```
.. figure:: /static/common/logo.png
   :width: 10 em
```

which returns

Replacement

You can put an image inside text or add an alias to use everywhere. To use an image inside a paragraph, first create an alias in the `source/substitutions.txt` file:

```
.. |nice_logo| image:: /static/common/logo.png
               :width: 1 em
```

and then call it in your paragraph:

```
My paragraph begins here with a nice logo |nice_logo|.
```

This is how the example will be displayed:

My paragraph begins here with a nice logo  .

To allow preview rendering in GitHub that is as close as possible to HTML rendering, you will also need to add the image replacement call at the end of the file you changed. This can be done by copy-pasting it from `substitutions.txt` or by executing the `scripts/find_set_subst.py` script.

Note

Currently, to ensure consistency and help in the use of QGIS icons, a list of aliases is built and available in the *Substitutions* chapter.

Figure

```
.. _figure_logo:

.. figure:: /static/common/logo.png
   :width: 20 em
   :align: center

   A caption: A logo I like
```

The result looks like this:

Fig. 2.1: A caption: A logo I like

To avoid conflicts with other references, always begin figure anchors with `_figure_` and use terms that easily connect to the figure caption. While only the centered alignment is mandatory for the image, feel free to use any other options for figures (such as `width`, `height`, `scale`...) if needed.

The scripts will insert an automatically generated number before the caption of the figure in the generated HTML and PDF versions of the documentation.

To use a caption (*see My caption*) just insert indented text after a blank line in the figure block.

A figure can be referenced using the reference label like this:

```
see :numref:`figure_logo`
```

renders like this:

see Fig. 2.1

This is the preferred way of referencing figures.

> **Note**
>
> For `:numref:` to work, the figure **must have a caption**.

It is possible to use `:ref:` instead of `:numref:` for reference, but this returns the full caption of the image.

```
see :ref:`figure_logo`
```

renders like this:

see *A caption: A logo I like*

2.1.7 Tables

You can make a simple table like this:

```
=======  =======  =======
x        y        z
=======  =======  =======
1        2        3
4                 5
=======  =======  =======
```

It will render like this:

x	y	z
1	2	3
4		5

Tables should have a caption. You can use an explicit reST "table" directive to add a caption to simple tables or grid tables. Add the caption on the same line as the directive and indent the table at least one space.

You can also add a hyperlink target before a table in order to reference it elsewhere. To avoid conflicts with other references, always begin hyperlink targets with `_table_` and use terms relevant to the table caption.

Here is an example of a more complicated grid table with a caption and a hyperlink target:

```
.. _table-grid-caption:

.. table:: Grid table with caption

   +---------------+--------------------+
   | Windows       | macOS              |
   +---------------+--------------------+
   | |win|         | |osx|              |
   +---------------+--------------------+
   | and of course not to forget |nix|  |
   +-----------------------------------+
```

The result:

Table 2.1: Grid table with caption

Windows	macOS
	
and of course not to forget 	

You may find it easier to use list tables or CSV tables to make complex tables. Add the caption after the `list-table` or `csv-table` directive.

Here is an example of a list table with a caption and a hyperlink target:

```
.. _table-list-caption:

.. list-table:: List table with caption
   :header-rows: 1
   :widths: 20 20 20 40

   * - What
     - Purpose
     - Key word
     - Description
   * - **Test**
     - ``Useful test``
     - complexity
     - Geometry.  One of:

       * Point
       * Line
```

The result:

Table 2.2: List table with caption

What	Purpose	Key word	Description
Test	`Useful test`	complexity	Geometry. One of: • Point • Line

Use `:numref:` roles to reference tables like this:

```
see :numref:`table-grid-caption` or :numref:`table-list-caption`
```

The result:

see Table 2.1 or Table 2.2

Note

You must add a caption to your table in order to create a cross reference with the `:numref:` role.

2.1.8 Index

An index is a handy way to help the reader find information in a document. QGIS documentation provides some essential indices. There are a few rules that help us provide a set of indices that are really useful (coherent, consistent and really connected to each other):

- An index should be human readable, understandable and translatable; an index can be made from many words but you should avoid any unneeded _, -… characters to link them i.e., `Loading layers` instead of `loading_layers` or `loadingLayers`.
- Capitalize only the first letter of the index unless the word has a particular spelling. E.g., `Loading layers`, `Atlas generation`, `WMS`, `pgsql2shp`.
- Keep an eye on the existing Index list in order to reuse the most convenient expression with the right spelling and avoid unnecessary duplicates.

Several index tags exist in reST. You can use the inline `:index:` tag within normal text:

```
QGIS can load several :index:`Vector formats` supported by GDAL ...
```

Or you can use the `.. index::` block-level markup which links to the beginning of the next paragraph. Because of the rules mentioned above, it is recommended to use the block-level tag:

```
.. index:: WMS, WFS, Loading layers
```

It is also recommended to use index parameters such as `single`, `pair` and `see`, in order to build a more structured and interconnected index table. See Index generating for more information on index creation.

2.1.9 Special Comments

Sometimes, you may want to emphasize some points of the description, either to warn, remind or give some hints to the user. In QGIS Documentation, we use reST special directives such as `.. warning::`, `.. seealso::`, `.. note::` and `.. tip::`. These directives generate frames that highlight your comments. See Paragraph Level markup for more information. A clear and appropriate title is required for both warnings and tips.

```
.. tip:: **Always use a meaningful title for tips**

   Begin tips with a title that summarizes what it is about. This helps
   users to quickly overview the message you want to give them, and
   decide on its relevance.
```

2.1.10 Code Snippets

You may also want to give examples and insert code snippets. In this case, write the comment below a line with the `::` directive inserted. For a better rendering, especially to apply color highlighting to code according to its language, use the code-block directive, e.g. `.. code-block:: xml`. More details at Showing code.

> **Note**
>
> Text in special comments will be translated, but text in code-block frames will not be translated. So keep comments in code blocks as short as possible and avoid comments unrelated to code.

2.1.11 Footnotes

This is for creating a footnote (showing as example[1])

```
blabla [1]_
```

Which will point to:

[1] Updates of core plugins

2.2 Managing Screenshots

2.2.1 Add new Screenshots

Here are some hints to create new, nice looking screenshots. The images should be placed in an image (`img/`) folder that is located in the same folder as the referencing `.rst` file.

- You can find some prepared QGIS projects that are used to create screenshots in the `./qgis-projects` folder of this repository. This makes it easier to reproduce screenshots for the next version of QGIS. These projects use the QGIS Sample Data (aka Alaska Dataset), which should be unzipped and placed in the same folder as the QGIS-Documentation Repository.
- Reduce the window to the minimal space needed to show the feature (taking the whole screen for a small modal window > overkill)
- The less clutter, the better (no need to activate all the toolbars)
- Don't resize them in an image editor; the size will be set into the `.rst` files if necessary (downscaling the dimensions without properly upping the resolution > ugly)
- Cut the background
- Make the top corners transparent if the background is not white
- Set print size resolution to `135 dpi` (e.g., in GIMP scale down the image using *Image* ► *Scale Image* and setting "X/Y" to `135 pixels/in`, and export it through *File* ► *Export...*). This way, images will be at original size in HTML and at a good print resolution in the PDF.

  You can also use ImageMagick convert command to do a batch of images:

  ```
  convert -units PixelsPerInch input.png -density 135 output.png
  ```

- Save them as `.png` (to avoid `.jpeg` artifacts)
- The screenshot should show the content according to what is described in the text

Tip

If you are on Ubuntu, you can use the following command to remove the global menu function and create smaller application screens with menus:

```
sudo apt autoremove appmenu-gtk appmenu-gtk3 appmenu-qt
```

2.2.2 Translated Screenshots

Here are some additional hints for those that want to create screenshots for a translated user guide:

Translated images should be placed in a `img/<your_language>/` folder. Use the same filename as the 'original' English screenshot.

2.3 Documenting Processing algorithms

If you want to write documentation for Processing algorithms, consider these guidelines:

- Processing algorithm help files are part of QGIS User Guide, so use the same formatting as the User Guide and other documentation.
- Each algorithm documentation should be placed in the corresponding **provider** folder and **group** file, e.g. the algorithm *Voronoi polygon* belongs to the *QGIS* provider and to the group *vectorgeometry*. So the correct file to add the description is: `source/docs/user_manual/processing_algs/qgis/vectorgeometry.rst`.

> **Note**
>
> Before starting to write the guide, check if the algorithm is already described. In this case, you can enhance the existing description.

- It is **extremely** important that each algorithm has an *anchor* that corresponds to the provider name + the unique name of the algorithm itself. This allows the Help button to open the Help page of the correct section. The anchor should be placed **above** the title, e.g. (see also the *Labels/references* section):

```
.. _qgisvoronoipolygons:

Voronoi polygons
----------------
```

  To find out the algorithm name you can just hover the mouse on the algorithm in the Processing toolbox.

- Avoid using "This algorithm does this and that…" as the first sentence in the algorithm description. Try to use more general expressions like:

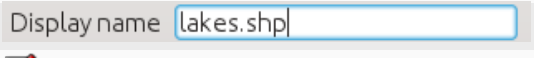
```
Takes a point layer and generates a polygon layer containing the...
```

- Avoid describing what the algorithm does by replicating its name and please don't replicate the name of the parameter in the description of the parameter itself. For example if the algorithm is `Voronoi polygon` consider to describe the `Input layer` as `Layer to calculate the polygon from`.
- Indicate in the description whether the algorithm has a default shortcut in QGIS or supports in-place editing.
- Add images! A picture is worth a thousand words! Use `.png` format and follow the general guidelines for documentation (see the *Figures and Images* section for more info). Put the image file in the correct folder, i.e. the `img` folder next to the `.rst` file you are editing.
- If necessary, add links in the "See also" section that provide additional information about the algorithm (e.g., publications or web-pages). Only add the "See also" section if there is really something to see. As a good practice, the "See also" section can be filled with links to similar algorithms.
- Give clear explanation for algorithm parameters and outputs: take inspiration from existing algorithms.
- Avoid duplicating detailed description of algorithm options. Add this information in the parameter description.
- Avoid adding information about the vector geometry type in the algorithm or parameter description, as this information is already available in the parameter descriptions.
- Add the default value of the parameter, e.g.:

```
* - **Number of points**
  - ``NUMBER_OF_POINTS``
  - [numeric: integer]

    Default: 1
  - Number of points to create
```

- Describe the *type* of input supported the parameters. There are several types available you can pick one from:

Parameter/Output type	Description	Visual indicator
Point vector layer	`vector: point`	
Line vector layer	`vector: line`	
Polygon vector layer	`vector: polygon`	
All spatial vector layers	`vector: geometry`	
Geometryless vector layer	`vector: table`	
Generic vector layer	`vector: any`	
Vector field numeric	`tablefield: numeric`	1.2
Vector field string	`tablefield: string`	abc
Vector field generic	`tablefield: any`	
Raster layer	`raster`	
Raster band	`raster band`	
HTML file	`html`	
Expression	`expression`	
Point geometry	`coordinates`	
Extent	`extent`	
CRS	`crs`	
Enumeration	`enumeration`	
List	`list`	
Integer value	`numeric: integer`	
Decimal value	`numeric: double`	
String	`string`	
Boolean	`boolean`	
Folder path	`folder`	
File	`file`	
Matrix	`matrix`	
Layer	`layer`	
Same output type as input type	`same as input`	
Definition	`definition`	
Point	`point`	
Map layers	`layer list`	
Range	`range`	
AuthConfig	`authconfig`	
Mesh	`mesh`	
Layout	`layout`	
LayoutItem	`layoutitem`	
Color	`color`	
Scale	`scale`	
Map theme	`map theme`	

- Study an existing and well documented algorithm, and copy all the useful layouts.
- When you are finished, just follow the guidelines described in *A Step By Step Contribution* to commit your changes and make a Pull Request

Here is an example of an existing algorithm to help you with the layout and the description:

```
.. _qgiscountpointsinpolygon:

Count points in polygon
```

(continues on next page)

(continued from previous page)

```
----------------------
Takes a point and a polygon layer and counts the number of points from the
point layer in each of the polygons of the polygon layer.
A new polygon layer is generated, with the exact same content as the input
polygon layer, but containing an additional field with the points count
corresponding to each polygon.

.. figure:: img/count_points_polygon.png
  :align: center

  The labels in the polygons show the point count

An optional weight field can be used to assign weights to each point.
Alternatively, a unique class field can be specified. If both options
are used, the weight field will take precedence and the unique class field
will be ignored.

``Default menu``: :menuselection:`Vector --> Analysis Tools`

Parameters
..........

.. list-table::
   :header-rows: 1
   :widths: 20 20 20 40

   * - Label
     - Name
     - Type
     - Description
   * - **Polygons**
     - ``POLYGONS``
     - [vector: polygon]
     - Polygon layer whose features are associated with the count of
       points they contain
   * - **Points**
     - ``POINTS``
     - [vector: point]
     - Point layer with features to count
   * - **Weight field**

       Optional
     - ``WEIGHT``
     - [tablefield: numeric]
     - A field from the point layer.
       The count generated will be the sum of the weight field of the
       points contained by the polygon.
   * - **Class field**

       Optional
     - ``CLASSFIELD``
     - [tablefield: any]
     - Points are classified based on the selected attribute and if
       several points with the same attribute value are within the
       polygon, only one of them is counted.
       The final count of the points in a polygon is, therefore, the
       count of different classes that are found in it.
   * - **Count field name**
     - ``FIELD``
     - [string]
```

(continues on next page)

(continued from previous page)

```
      Default: 'NUMPOINTS'
    - The name of the field to store the count of points
  * - **Count**
    - ``OUTPUT``
    - [vector: polygon]

      Default: [Create temporary layer]
    - Specification of the output layer type (temporary, file,
      GeoPackage or PostGIS table).
      Encoding can also be specified.

Outputs
.......

.. list-table::
   :header-rows: 1
   :widths: 20 20 20 40

   * - Label
     - Name
     - Type
     - Description
   * - **Count**
     - ``OUTPUT``
     - [vector: polygon]
     - Resulting layer with the attribute table containing the
       new column with the points count
```

CHAPTER

THREE

WRITING CODE IN THE PYQGIS COOKBOOK

If you are planning to add or update some chapters of the PyQGIS-Developer-Cookbook, then you should follow some rules to enable automatic testing of the code snippets.

Testing is really important because it allows automatic checking of the code. Code snippets with errors or code that uses outdated methods will fail and the notification will help you fix the problems.

For testing, we use the Sphinx doctest extension. Refer to the extension documentation for more detailed information.

3.1 How to write testable code snippets

Writing testable code snippets is not so different from the *old* method. Basically, you need to use a different Sphinx *directive*.

3.1.1 Doctest sphinx directives

Instead of embedding the code in a `.. code-block:: python` directive (which would highlight the code syntax automatically), you now need to embed it in a `.. testcode::`. That is, instead of this:

```
.. code-block:: python

   crs = QgsCoordinateReferenceSystem("EPSG:4326")
   assert crs.isValid()
```

You now use this:

```
.. testcode::

   crs = QgsCoordinateReferenceSystem("EPSG:4326")
   assert crs.isValid()
```

After you wrote the example code, you should add some *assertion* that will evaluate the code and that will be run automatically.

In the above example, you are creating a crs and with `assert crs.isValid()` you **test** if it is valid. If the code has a wrong python syntax or the `crs.isValid()` returns `False`, this code snippet will fail during testing.

To successfully run the tests on snippets, you must import all the classes and declare any variables used in the code snippets. You can include those in the code snippet itself (visible in the HTML pages) or you can add them to a `.. testsetup::` directive (hidden in the HTML pages). The `.. testsetup::` needs to be placed before the `.. testcode:::`

```
.. testsetup::

   from qgis.core import QgsCoordinateReferenceSystem

.. testcode::

   crs = QgsCoordinateReferenceSystem("EPSG:4326")
   assert crs.isValid()
```

If the code snippet doesn't create objects (and therefore you cannot use something like `assert object.isValid()`), you can test the code using the `print()` method, then add the expected results within a `.. testoutput::` directive to compare the expected output:

```
.. testcode::

   print("QGIS CRS ID:", crs.srsid())
   print("PostGIS SRID:", crs.postgisSrid())

.. testoutput::

   QGIS CRS ID: 3452
   PostGIS SRID: 4326
```

By default, the content of `.. testoutput::` is shown in the HTML output. To hide it from the HTML use the *:hide:* option:

```
.. testoutput::
   :hide:

   QGIS CRS ID: 3452
   PostGIS SRID: 4326
```

Note

If the code snippet contains any print statements, you MUST add a `testoutput` with the expected outputs; otherwise the test will fail.

3.1.2 Grouping tests

For each reST document, the code snippets are tested sequentially, which means you can use one `.. testsetup::` for all the following code snippets and that later snippets will have access to variables declared in earlier ones in the document.

Alternatively, you can use groups to break down the examples on the same page in different tests.

You add the code snippet to groups by adding one or more group names (separated by commas) in the respective directive:

```
.. testcode:: crs_crsfromID [, morenames]

   crs = QgsCoordinateReferenceSystem("EPSG:4326")
   assert crs.isValid()
```

The `doctest` will pick each group snippets and run them independently.

Note

Use group names that make sense with the related content. Use something similar to <chapter>_<subchapter>, for example: crs_intro, crs_fromwkt. In case of failures, this will help identifying where the failures occur.

If you don't declare any group, the code snippet will be added to a group named `default`. If instead, you use `*` as a group name, the snippet will be used in all testing groups, something normally useful to use in the test setup:

```
.. testsetup:: *

   from qgis.core import QgsCoordinateReferenceSystem
```

3.2 How to test snippets on your local machine

Note

Instructions are valid for Linux system.

To test Python code snippets, you need a *QGIS* installation. For this, there are many options. You can:

- Use your system *QGIS* installation with *Sphinx* from a Python virtual environment:

  ```
  make -f venv.mk doctest
  ```

- Use a manually built installation of *QGIS*. You'd need to:

  1. Create a custom `Makefile` extension on top of the `venv.mk` file, for example a `user.mk` file with the following content:

     ```
     # Root installation folder
     QGIS_PREFIX_PATH = /home/user/apps/qgis-master

     include venv.mk
     ```

     Or

     ```
     # build output folder
     QGIS_PREFIX_PATH = /home/user/dev/QGIS-build-master/output

     include venv.mk
     ```

  2. Then, use it to run target `doctest`:

     ```
     make -f user.mk doctest
     ```

- Run target `doctest` inside the official *QGIS* docker image:

  ```
  make -f docker.mk doctest
  ```

  You have to install Docker first because this uses a docker image with QGIS in it.

CHAPTER

FOUR

TRANSLATION GUIDELINES

This manual is aiming to help the translator. First, it explains how you can join the translation team. Then the general process of how technically a translation is done is explained. Later the translation is explained from an actual English `.rst` document that is translated to Dutch. Finally a summary of *rules of translation* is given.

Note

Although these guidelines focus on QGIS documentation, the methods and the rules described below are also applicable to QGIS applications.

4.1 Becoming a translator

The QGIS project is always looking for people who are willing to invest some more time translating QGIS into a foreign language - even perhaps to coordinate the translation effort.

We are trying to improve our project management process and spread the load more evenly between people who each have a specific area of responsibility, so any contribution you have to make will be greatly appreciated.

If you would like to nominate yourself as a coordinator for a new language please go ahead. If more than one person nominate themselves as coordinator for the same language, please contact each other and resolve how you will manage your efforts.

4.1.1 Transifex

The web-based translating platform Transifex is used for all QGIS translations: the desktop application itself (or GUI) and the documentation. So the first thing you need is an account to login and get started.

4.1.2 Join a Project

1. Go to https://explore.transifex.com/qgis/
2. You can explore available projects we translate, identify their target languages with various statistics:
   - **QGIS Desktop** for all the pieces of text available in QGIS apps (QGIS Desktop and QGIS Server),
   - **QGIS Documentation** for the official LTR documentation
3. Click on the project you would like to help translate
4. Click on *JOIN THIS PROJECT* on the right side. You will be prompted to sign up.
5. Create your account or connect using a third-party platform account. Verify your account by the link in the email you will receive.
6. Login
7. You then get a popup to select which language you want to help translate to. Please, note that we try to make the process as simple as possible and only mention target languages, regardless of the country parameter (e.g. `French (fr)` and **NOT** `French (France) (fr_FR)`). Only if there are notable differences in the languages (e.g. portuguese in Portugal vs Brazil) we may allow different versions.

   Search your target language, i.e the language you wish to help translate QGIS into, NOT necessarily all the languages you can speak:

   - If it is marked as `already added` then select it and press *Join Project*.
   - If it is not marked as already added, select it and press *Request language*. Keep in mind that translating an entire project will take days of work, if not weeks! Again, and sorry to repeat, it is not about selecting ALL the languages you can speak.
8. Now you will need to wait for the language coordinator or the project maintainers to process your request. You will be notified by email when your request has been accepted. If your request has no answer for about a week, please consider writing to your language coordinator in Transifex or the QGIS Translators mailing list.
9. You can also join any of the other QGIS projects and help everywhere too.

4.1.3 Translate

Once your request is accepted, you are able to translate any text in the project(s) you've chosen. Simply click on your language, select the chapter you want to translate and click on Translate. Easy, right?

In order to help you make good translation, some instructions are provided below. We strongly recommend you to read them.

> **Tip**
>
> **Quick access to translatable files in Transifex**
>
> If you find a wrong or missing translation in the current documentation, you can use the `Translate page` link in the bottom left drop-down menu of the page to reach it sources in Transifex and perform any update you wish to.

4.2 Translation process

QGIS Documentation is written in English with `.rst` files. In order to provide translations:

1. A prebuild script creates translation files named `.po` files for the English language in the folder `/QGIS-Documentation/locale/en`.
2. The sentences in the `.po` files are pushed to the Transifex web platform, and made available for translators who can begin to translate from English to their language with the editor.
3. When a file is translated at 100%, the translated strings are automatically pulled back to the documentation repository, under `/QGIS-Documentation/locale/<language>`.
4. At the next build of the documentation (which occurs at least once a day – see time at the bottom of the page), a script reuses the sentences to create translated output.
5. For files not fully translated, a script pulls every two weeks translated strings from Transifex to Github and these are as well published at the next build.
6. Whenever an `.rst` file is updated, the English `.po` file is updated and the changes are pushed to the corresponding file in Transifex. This means that when a new paragraph is added to an `.rst` document that was already translated, only the new/updated sentences are added to the translated `.po` file and needs to be translated.

Note

Translating QGIS Desktop specificities

The main difference with translating QGIS applications is that instead of `.po` files, all the translatable strings in the `.py`, `.cpp`, `.yaml` files that shape a particular version of the application are pushed to and pulled from Transifex as a single `.ts` file (e.g. `qgis-application/qgis_en.ts (branch release-3_30)`). Translations are pulled to Github in development branch (daily), and at release time (for every released versions).

Two different tools are currently used to do translations in QGIS:

- The Transifex web platform, the **easiest and recommended way** to translate QGIS, transparently does the process described above and pulls all the translatable texts in one place for the translator. Just pick the files you want and translate. Translated files are stored in the platform until another release is pushed.
- Qt Linguist, a Qt development tool, requires the translator to pull locally the `.po` (or `.ts`) files from the source code, translate and then push back.

Note that whatever tool you choose, rules of translations are the same.

4.3 Translate a file

To explain how translation works, we will use the heatmap plugin as an example. In this example we will translate it from English to Dutch, but it will be practically the same for other documents in all languages.

The source of the document can be found here:

```
QGIS-Documentation/source/docs/user_manual/plugins/plugins_heatmap.rst
```

So why did I choose this document?

1. It includes images, captions, headers, references and replacements.
2. I wrote it so it is easier for me to translate ;-)

The build process has created the English `.po` file which can be found here:

```
QGIS-Documentation/locale/en/LC_MESSAGES/docs/user_manual/plugins/plugins_heatmap.
↪po
```

The equivalent Dutch `.po` file (basically a copy) can be found here:

```
QGIS-Documentation/locale/nl/LC_MESSAGES/docs/user_manual/plugins/plugins_heatmap.
↪po
```

Along this file you will see a tiny `.mo` file which indicates that it does not hold any translations yet.

4.3.1 Translation in Transifex

In order to translate using Transifex, you need to:

1. *create an account on Transifex and join the QGIS project*.
2. Once you are part of a language team, click on the corresponding project (in this case `QGIS Documentation`). A list of available languages with their ratio of translation is displayed.

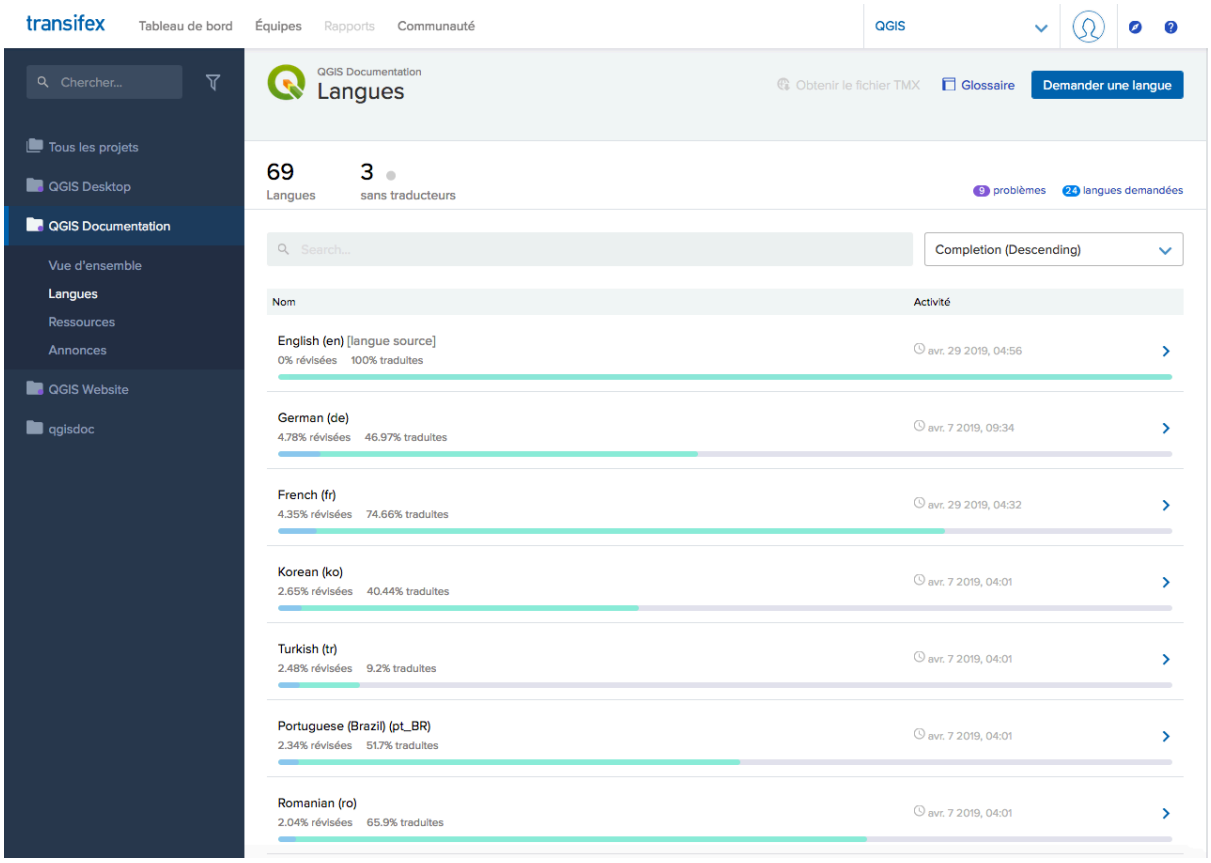


Fig. 4.1: Select language for translation in the Transifex menu

3. Hover over your language and click either:
   - *View resources*: translatable `.po` files with their ratio of translation, number of strings and some more metadata are now displayed.
   - or *Translate*: opens the interface of translation with all the available `.po` files
4. Identify the file you'd like to translate (in our case we are looking for the `docs_user-manual_plugins_plugins-heatmap`, the heatmap plugin file) or any unfinished

file and click on it: strings in the files are loaded and you can use the interface to filter, translate, suggest translation…

> **Tip**
>
> Clicking the `Translate page` link in the bottom left drop-down menu of a page brings you directly to its corresponding translation page in Transifex.

5. All you need to do is select each text and translate following the *guidelines*.

For further information on the use of Transifex Web Editor, see https://help.transifex.com/en/articles/6318216-translating-with-the-web-editor.

4.3.2 Translation in Qt Linguist

With Qt Linguist, you need to:

1. manually grab the `.po` or `.ts` file(s). This can be achieved by downloading the file(s) either from Transifex platform or from the `locale/$language` folder of the source repository (in GitHub),
2. proceed to the translation locally
3. upload the modified files to their sources (Transifex or GitHub).

While downloading and uploading translatable files can be done with Transifex, it's not advised to use this process. Since there's no versioning system on Transifex, the file you upload will simply replace the existing one and potentially overwrite any modification made by others on the platform in the meantime.

When you open the file in Qt Linguist for the first time you will see the following dialog:

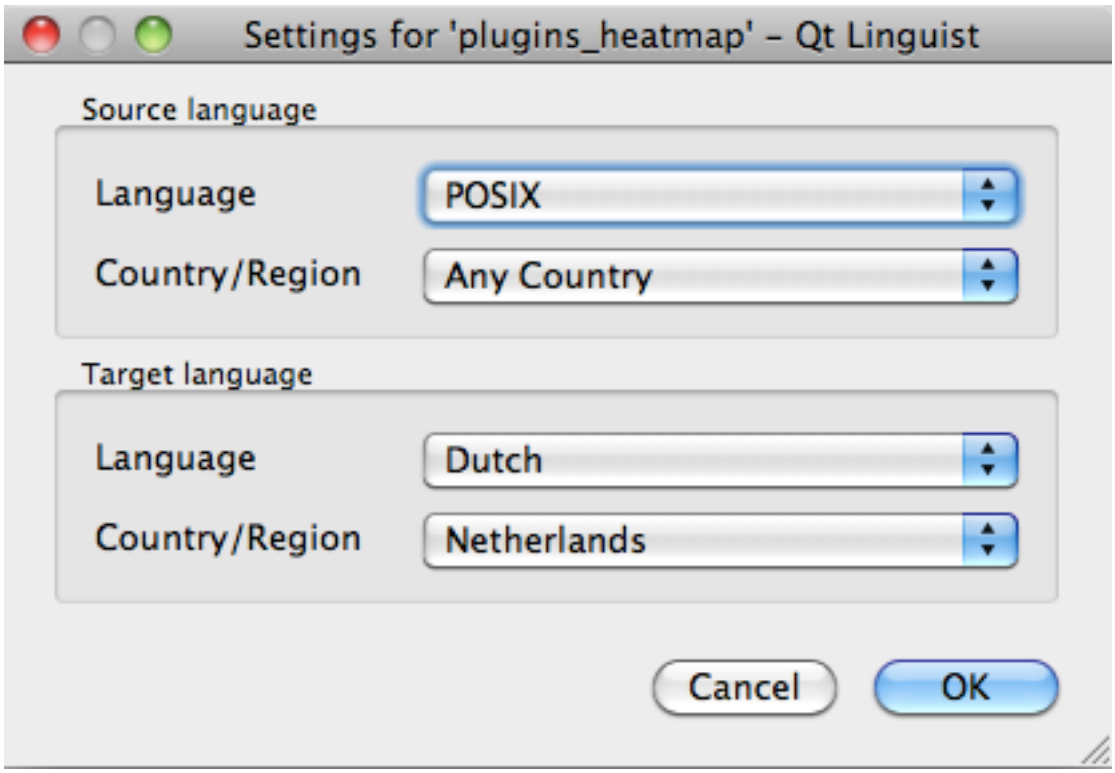


Fig. 4.2: Select language for translation in linguist menu

The Target language should be filled correctly. The Source language can be left as is with language POSIX and Country/Region on Any Country.

When you press the *OK* button Qt Linguist is filled with sentences and you can start translating, see Fig. 4.3.

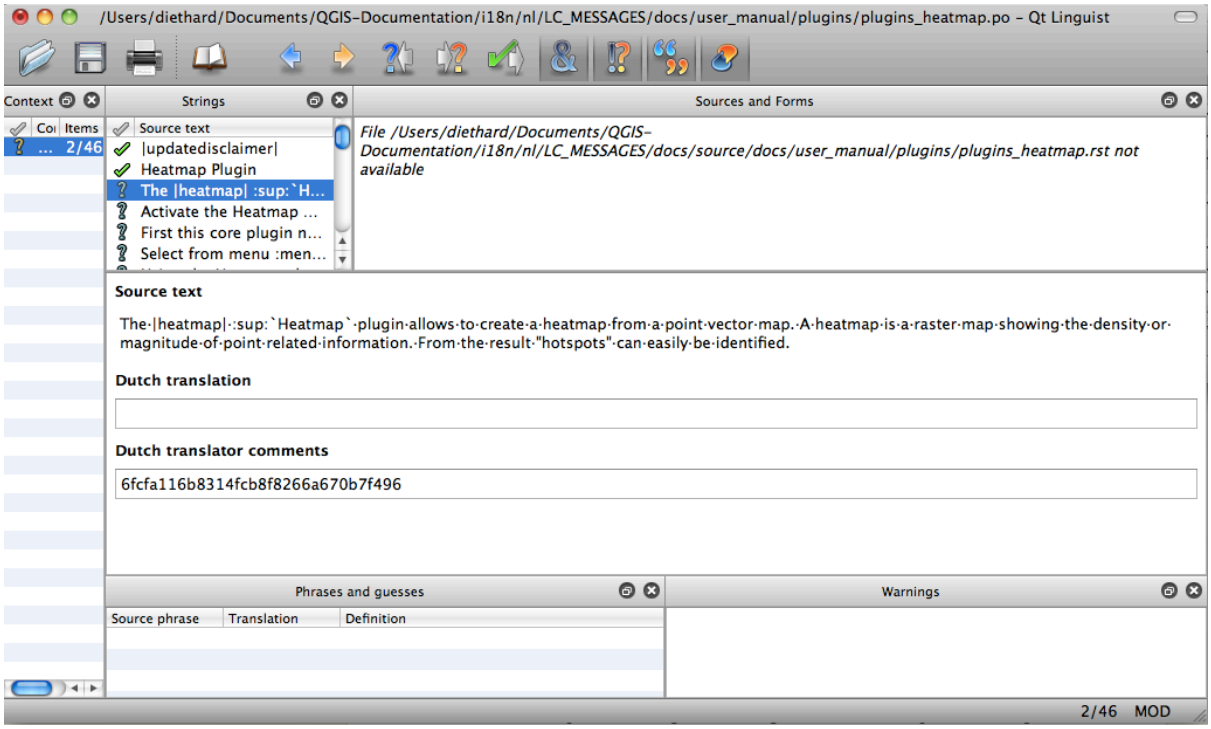


Fig. 4.3: Translate using the linguist menu

In the menu you see the following buttons which are convenient to use.

-  The Translation Done Next button, is the most important button. If the item needs translation, you enter a translation in the text field, then hit this button. If the item does not need translation just leave the text field for translation empty and also hit this button which indicates the item is done and you continue with the next item.
-  The Goto Previous button, can be used to go to the previous translation item.
-  The Goto Next button, can be used to go to the next translation item.
-  The Next Todo button, jumps to the first translation item that still needs a translation. Handy when the original document has changed and only several new/changed sentences need to be translated.
-  The Previous Todo button, searches backward and jumps to the first translation item it finds that still needs a translation.

For further information on the use of Qt Linguist, see https://doc.qt.io/qt-5/linguist-translators.html

Warning

If you want to download content to translate from the source repository, never do this in the `master` branch. For translations there are always translation branches available, once a document is fully updated in English for a certain version. As an example, to translate the manual of QGIS 2.8, you have to use the manual_en_v2.8 branch.

4.3.3 Translate a manual

Now we start to translate the plugin_heatmap manual!

Translating most of the sentences should be straightforward. During this translation session I will point out which parts (rst statements) need special translation.

Below we see an interesting sentence to translate:

```
The |heatmap| :sup:`Heatmap` plugin allows to create a heatmap from a
point vector map. A heatmap is a raster map showing the density or
magnitude of point related information. From the result "hotspots" can
easily be identified.
```

This sentence contains two rst statements:

1. `|heatmap|` words between `|` are replacements and these should never be translated! This will be replaced by the heatmap plugin icon!
2. `:sup:`Heatmap``, the `:sup:` statement is a superposition statement and prints the following text a bit higher. This is used to show the popup texts that appear when you hover above the toolbar item and this may be different when it is actually translated in the QGIS application. In the Dutch case it is not!

All other plain text in this sentence can be translated!

The next translation item contains the `:ref:` statement that is commonly used to refer to another section somewhere in the manual! The text following a `:ref:` statement should never be changed because it is a unique identifier!

```
First this core plugin needs to be activated using the Plugin Manager
(see Section :ref:`load_core_plugin`). After activation the heatmap icon
|heatmap| can be found in the Raster Toolbar.
```

In this case `load_core_plugin` is a unique reference identifier placed before an rst item that has a caption. The ref statement will be replaced with the text of the header and turned into a hyperlink. When the header this reference is referring to is translated, all references to this header will be automatically translated as well.

The next item contains the rst-tag `:menuselection:` followed by text actually displayed in a menu in QGIS application, this may be translated in the application and therefore should be changed when this is the case.

```
Select from menu :menuselection:`View --> Toolbars --> Raster` to activate
the Raster Toolbar when it is not yet activated.
```

In above item "View –>" is actually translated to "Beeld –>" because this is the translation used in the Dutch localized QGIS application.

A bit further we meet the following tricky translation item:

```
The |heatmap| :sup:`Heatmap` tool button starts the Dialog of the Heatmap
plugin (see :numref:`figure_heatmap_settings`).
```

It holds a reference to a figure `figure_heatmap_settings_`, and like a reference to a section this reference should not be changed!! The reference definition from the rst-document is not included in the `.po` file and can therefore not be changed. This means the reference to figures can not be translated. When HTML is created you will see `figure_heatmap_settings`. When a PDF document is created `figure_heatmap_settings_` is replaced with a figure number.

The next translation item with rst attributes is the following item:

```
**Input Point dialog**: Provides a selection of loaded point vector maps.
```

Do not remove the stars in above line. It will print the text it holds in bold. The text itself is often text included in the dialog itself and may well be translated in the application.

The following translation item contains the `:guilabel:` rst tag.

```
When the |checkbox| :guilabel:`Advanced` checkbox is checked it will
give access to additional advanced options.
```

The text `Advanced` of the guilabel tag may well be translated in the QGIS application and probably needs to be changed!

The following translation item contains ``airports``. The quotes are used to give the text another text font. In this case it is a literal value and does not need translation.

```
For the following example, we will use the ``airports`` vector point
layer from the QGIS sample dataset (see :ref:`label_sampledata`).
Another excellent QGIS tutorial on making heatmaps can be found on
`https://www.qgistutorials.com
<https://www.qgistutorials.com/en/docs/creating_heatmaps.html>`_.
```

This item also includes a hyperlink with an url and an external presentation. The url should of course be left intact, you are allowed to change the external text `https://www.qgistutorials.com` which is visible by the reader. Never remove the underscore at the end of the hyperlink which forms an essential part of it!!

4.3.4 Summary Rules for translation

1. Do not change text between two `|` characters like `|bronze|`, `|checkbox|`, `|labels|`, `|selectString|`, `|addLayer|` … These are special tags used to replace images
2. Do not change references that start with roles like `:ref:`, `:file:`, `:numref:` unless they include a title. In that case, you can translate the title but keep unchanged the link (i.e., the text between < and >)

   > **Tip**
   >
   > When a title is provided for a reference, Transifex may display a number in the English source text in replacement of the link part. Click on the number in the source text to add the reference link next to the title being translated.

3. Do not change references that end with an underscore like `figure_labels_1_`
4. Do not change the url in hyperlinks, but you may change the external description. Leave the underscore at the end of the hyperlink, without additional spacing (>`_)
5. Change the text inside quotes following `:index:`, `:sup:`, `:guilabel:` and `:menuselection:` tags. Check if/how it is translated in the QGIS Application. Do not change the tag itself.
6. Text between double stars and double quotes often indicate values or fieldnames, sometimes they need translation sometimes not.
7. Be aware to use exactly the same (number of) special characters of the source text such as `, ``, *, **, ::. These contribute to the cosmetics of the information provided
8. Do not begin nor end the text hold by special characters or tags with a space
9. Do not end the translated strings with a new paragraph, otherwise the text will not be translated during the html generation.

Stick to above presented rules and the translated document will look fine!

For any question, please contact the QGIS Community Team or the QGIS Translation Team.

CHAPTER

FIVE

SUBSTITUTIONS

5.1 Usage

To ease the use of icons in QGIS manuals, replacements are defined for each icon in `/source/substitutions.txt` file at QGIS-Documentation repository and some of these substitutions are listed below. Thus, when you want to use an icon from QGIS application in the documentation there is a big chance that there is already a substitution that can/should be used.

If no replacement exists:

1. check the documentation repository whether the icon is available in `/static/common` folder. If no image, then you need to find and copy the icon image file from QGIS repository (often under default themes folder) and paste (in `.png` format) under `/static/common` folder. For convenience and update, it's advised to keep filename when possible.
2. create the reference to the substitution in the `/substitutions.txt` file following the example below. The replacement text should be derived from file name and in camelCase:

```
.. |dataSourceManager| image:: /static/common/mActionDataSourceManager.png
   :width: 1.5em
.. |splitLayer| image:: /static/common/split_layer.png
   :width: 1.5em
```

3. Update the target section(s) of the docs, using your new substitution.
4. (optional but highly desirable) add the substitution to the list below.
5. Add the new substitution reference in the substitutions list at the end of the file(s) it is used in, or run the convenient `scripts/find_set_subst.py` script.

```
# from the repository main folder
python3 scripts/find_set_subst.py
```

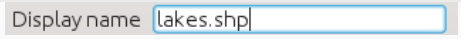
5.2 Common Substitutions

Below are given some icons and their substitution to use when writing documentation. Can be used/found in many places in manuals.

5.2.1 Platform Icons

Icon	Substitution	Icon	Substitution
	`|logo|`		`|qt|`
	`|kde|`		`|nix|`
	`|osx|`		`|win|`

5.2.2 Menu Items

Icon	Substitution	Icon	Substitution
	|checkbox|		|unchecked|
	|radioButtonOn|		|radioButtonOff|
	|selectNumber|		|selectString|
	|selectColor|		|selectColorRamp|
	|tab|	°	|degrees|
	|inputText|		|slider|
	|hamburgerMenu|		

5.3 Toolbar Button Icons

5.3.1 Manage Layers and overview

Icon	Substitution	Icon	Substitution
	|dataSourceManager|		
	|addOgrLayer|		|addSensorThingsLayer|
	|addRasterLayer|		|addMssqlLayer|
	|addDelimitedTextLayer|		|addSpatiaLiteLayer|
	|addPostgisLayer|		|addOracleLayer|
	|addAfsLayer|		|addMeshLayer|
	|addVectorTileLayer|		|addXyzLayer|
	|addVirtualLayer|		|addWmsLayer|
	|addWcsLayer|		|addWfsLayer|
	|addPointCloudLayer|		|addGpsLayer|
	|addTiledSceneLayer|		|addHanaLayer|
	|newVectorLayer|		|newSpatiaLiteLayer|
	|newGeoPackageLayer|		|createMemory|
	|newVirtualLayer|		|newMeshLayer|
	|newGpx|		|stac|
	|dbManager|		|gdal|
	|geoPackage|		|spatialite|
	|virtualLayer|		|wms|

continues on next page

Table 5.1 – continued from previous page

Icon	Substitution	Icon	Substitution
	|wcs|		|wfs|
	|pointCloudLayer|		|gps|
	|tiledSceneLayer|		|hana|
	|dbSchema|		
	|inOverview|		|addAllToOverview|
	|removeAllFromOverview|		|removeLayer|
	|showAllLayers|		|hideAllLayers|
	|showPresets|		|showSelectedLayers|
	|hideSelectedLayers|		|hideDeselectedLayers|
	|toggleAllLayers|		|toggleSelectedLayers|
	|addLayer|		
	|indicatorTemporal|		|indicatorNonRemovable|
	|indicatorEmbedded|		|indicatorFilter|
	|indicatorMemory|		|indicatorNoCRS|
	|indicatorBadLayer|		|favourites|
	|indicatorLayerError|		|indicatorNotes|
	|indicatorLowAccuracy|		|indicatorOffline|

5.3.2 Project

Icon	Substitution	Icon	Substitution
	|fileNew|		|fileOpen|
	|fileSave|		|fileSaveAs|
	|fileExit|		|user|

5.3.3 Edit

Icon	Substitution	Icon	Substitution
	|undo|		|redo|
	|editCopy|		|editPaste|
	|editCut|		|saveEdits|
	|editableEdits|		
	|circle2Points|		|circle2TangentsPoint|
	|circle3Points|		|circle3Tangents|
	|circleCenterPoint|		|ellipseCenter2Points|
	|ellipseCenterPoint|		|ellipseExtent|
	|ellipseFoci|		|rectangle3PointsDistance|
	|rectangle3PointsProjected|		|rectangleCenter|
	|rectangleExtent|		|regularPolygon2Points|
	|regularPolygonCenterCorner|		|regularPolygonCenterPoint|

5.3.4 Identify result

Icon	Substitution	Icon	Substitution
	|expandTree|		|collapseTree|
	|expandNewTree|		|formView|
	|deselectAll|		|editCopy|
	|filePrint|		
	|identifyByRectangle|		|identifyByPolygon|
	|identifyByFreehand|		|identifyByRadius|

5.3.5 Digitizing and Advanced Digitizing

Icon	Substitution	Icon	Substitution
	|cad|		|cadConstruction|
	|cadParallel|		|cadPerpendicular|
	|floater|		
	|toggleEditing|		|allEdits|
	|tracing|		|snapping|
	|snappingVertex|		|snappingSegment|
	|snappingArea|		|snappingCentroid|

continues on next page

Table 5.2 – continued from previous page

Icon	Substitution	Icon	Substitution
	|snappingMiddle|		|snappingEndpoint|
	|capturePoint|		|capturePolygon|
	|captureLine|		|captureCurveFromFeature|
	|deleteSelectedFeatures|		
	|circularStringCurvePoint|		|circularStringRadius|
	|vertexTool|		|vertexToolActiveLayer|
	|digitizeWithSegment|		|digitizeShape|
	|streamingDigitize|		|digitizeWithCurve|
	|moveFeature|		|moveFeatureCopy|
	|moveFeatureLine|		|moveFeatureCopyLine|
	|moveFeaturePoint|		|moveFeatureCopyPoint|
	|rotateFeature|		|rotatePointSymbols|
	|scaleFeature|		
	|offsetCurve|		|offsetPointSymbols|
	|simplify|		|reshape|
	|addRing|		|addPart|
	|fillRing|		
	|deleteRing|		|deletePart|
	|mergeFeatures|		|mergeFeatureAttributes|
	|splitFeatures|		|splitParts|
	|reverseLine|		
	|allowIntersections|		|avoidIntersectionsCurrentLayer|
	|avoidIntersectionsLayers|		|snappingSelf|

5.3.6 Mesh

Icon	Substitution	Icon	Substitution
	|meshDigitizing|		|meshReindex|
	|meshSelectExpression|		|meshSelectPolygon|
	|meshTransformByExpression|		|meshEditForceByVectorLines|
	|vertexCoordinates|		

5.3.7 Map Navigation and attributes

Icon	Substitution	Icon	Substitution
	|pan|		|panToSelected|
	|zoomIn|		|zoomOut|
	|zoomActual|		|zoomFullExtent|
	|zoomToLayer|		|zoomToSelected|
	|zoomLast|		|zoomNext|
	|zoomInXAxis|		|refresh|
	|identify|		|mapTips|
	|showBookmarks|		|newBookmark|
	|measure|		|measureArea|
	|measureBearing|		|measureAngle|
	|newMap|		|new3DMap|
	|tiltUp|		|tiltDown|
	|3dNavigation|		|play|
	|camera|		|shadow|
	|editCutDisabled|		
	|temporal		|temporalNavigationOff|
	|temporalNavigationFixedRange|		|temporalNavigationAnimated|
	|temporalNavigationMovie|		|newElevationProfile|
	|elevation|		

5.3.8 Selection and Expressions

Icon	Substitution	Icon	Substitution
	`|selectRectangle|`		`|selectPolygon|`
	`|selectFreehand|`		`|selectRadius|`
	`|selectAll|`		`|deselectAll|`
	`|invertSelection|`		`|expressionSelect|`
	`|deselectActiveLayer|`		
	`|selectDistance|`		`|selectLocation|`
	`|selectAllTree|`		`|select|`
	`|selectAdd|`		`|selectRemove|`
	`|formSelect|`		`|dataDefine|`
	`|expression|`		`|dataDefineOn|`
	`|dataDefineExpressionOn|`		`|dataDefineError|`
	`|dataDefineExpressionError|`		
	`|addExpression|`		
	`|expressionFilter|`		`|filterMap|`

5.3.9 Labels and Diagrams

Icon	Substitution	Icon	Substitution
	|labelingSingle|		|labelingNone|
	|labelingRuleBased|		|labelingObstacle|
	|piechart|		|diagramNone|
	|text|		|histogram|
	|stackedBar|		|stackedDiagram|
	|createAnnotationLayer|		|annotationLayer|
	|formAnnotation|		|htmlAnnotation|
	|actionText|		|textAlongLine|
	|addImage|		|textInsideRect|
	|labelbackground|		|labelbuffer|
	|labelformatting|		|labelplacement|
	|labelshadow|		|render|
	|labelcallout|		
	|labelAnchorCenter|		|labelAnchorCustom|
	|labelAnchorEnd|		|labelAnchorStart|
	|pinLabels|		|showHideLabels|
	|moveLabel|		|rotateLabel|
	|showPinnedLabels|		|showUnplacedLabel|
	|changeLabelProperties|		|autoPlacementSettings|
	|labelingRules|		

5.3.10 Decorations

Icon	Substitution	Icon	Substitution
	|copyrightLabel|		|addGrid|
	|titleLabel|		|northArrow|
	|scaleBar|		|addMap|
	|addImage|		

5.3.11 Help

Icon	Substitution	Icon	Substitution
	|helpContents|		|qgisHomePage|
	|success|		
	|helpSponsors|		|contextHelp|

5.3.12 Colors

Icon	Substitution	Icon	Substitution
	|colorBox|		|colorPicker|
	|colorSwatches|		|colorWheel|

5.4 Other basic icons

Icon	Substitution	Icon	Substitution
	|arrowLeft|		|arrowRight|
	|arrowDown|		|arrowUp|
	|symbologyAdd|		|symbologyRemove|
	|projectProperties|		|options|
	|interfaceCustomization|		|keyboardShortcuts|
	|copyrightLabel|		|northArrow|
	|scaleBar|		|tracking|
	|gpsTrackBarChart|		
	|gpsConnect|		|gpsDisconnect|
	|gpsDestinationLayer|		|addTrackPoint|
	|recenter|		|reset|
	|folder|		|extents|
	|settings|		|start|
	|properties|		|deleteSelected|
	|browserExpand|		|browserCollapse|
	|codeEditor|		|add|
	|relations|		|layoutItem3DMap|
	|stopwatch|		|sensor|
	|clearItem|		|warning|

5.5 Attribute Table

Icon	Substitution	Icon	Substitution
	|openTable|		|openTableSelected|
	|openTableVisible|		|openTableEdited|
	|selectedToTop|		
	|selectAll|		|invertSelection|
	|panToSelected|		|zoomToSelected|
	|copySelected|		|editPaste|
	|expressionSelect|		|deleteSelectedFea-tures|
	|newAttribute|		|deleteAttribute|
	|editTable|		
	|newTableRow|		|calculateField|
	|refresh|		|formView|
	|conditionalFormatting|		|multiEdit|
	|dock|		|actionRun|
	|duplicateFeature|		|zoomTo|
	|panTo|		|highlightFeature|
	|handleStoreFilterExpressionChecked|		
	|handleStoreFilterExpressio-nUnchecked|		

5.6 Projections and Georeferencer

Icon	Substitution	Icon	Substitution
	|geographic|		|crs|
	|customProjection|		|setProjection|
	|projectionDisabled|		|projectionEnabled|
	|transformation|		|gdalScript|
	|georefRun|		|pencil|
	|linkQGisToGeoref|		|linkGeorefToQGis|
	|fullHistogramStretch|		

5.7 Print Layout

Icon	Substitution	Icon	Substitution
	|newLayout|		|layoutManager|
	|duplicateLayout|		
	|newReport|		|newPage|
	|atlasSettings|		|atlas|
	|filePrint|		|saveMapAsImage|
	|saveAsSVG|		|saveAsPDF|
	|addBasicShape|		|addBasicCircle|
	|addBasicTriangle|		|addBasicRectangle|
	|addNodesShape|		|editNodesShape|
	|addPolygon|		|addPolyline|
	|addArrow|		|northArrow|
	|add3DMap|		|addMap|
	|elevationProfile|		|copyProfileSettings|
	|addLegend|		|addHtml|
	|addManualTable|		|addTable|
	|addImage|		|addMarker|
	|label|		|scaleBar|
	|select|		|moveItemContent|
	|setToCanvasScale|		|setToCanvasExtent|
	|viewScaleInCanvas|		|viewExtentInCanvas|
	|raiseItems|		|lowerItems|
	|moveItemsToTop|		|moveItemsToBottom|
	|alignLeft|		|alignRight|
	|alignHCenter|		|alignVCenter|
	|alignTop|		|alignBottom|
	|distributeLeft|		|distributeRight|
	|distributeTop|		|distributeBottom|
	|distributeHCenter|		|distributeVCenter|
	|distributeHSpace|		|distributeVSpace|
	|resizeShortest|		|resizeTallest|
	|resizeNarrowest|		|resizeWidest|
	|resizeSquare|		|groupItems|
	|lockItems|		|unlockAll|

continues on next page

Table 5.3 – continued from previous page

Icon	Substitution	Icon	Substitution
	|locked|		|unlocked|
	|lockRepeating|		|lockedGray|

5.8 Layer Properties

Icon	Substitution	Icon	Substitution
	|symbology|		|labelingSingle|
	|sourceFields|		|general|
	|metadata|		|action|
	|display|		|rendering|
	|join|		|diagram|
	|labelmask|		|temporal|
	|legend|		|dependencies|
	|3d|		|system|
	|elevationscale|		|layerTree|
	|editMetadata|		|overlay|
	|digitizing|		|auxiliaryStorage|
	|history|		|stylePreset|
	|search|		|pyramids|
	|transparency|		|rasterHistogram|
	|singleSymbol|		|nullSymbol|
	|graduatedSymbol|		|categorizedSymbol|
	|25dSymbol|		|ruleBasedSymbol|
	|invertedSymbol|		|heatmapSymbol|
	|pointDisplacementSymbol|		|pointClusterSymbol|
	|mergedFeatures|		
	|meshcontours|		|meshcontoursoff|
	|meshvectors|		|meshvectorsoff|
	|meshframe|		|meshaveraging|
	|singleColor|		|paletted|
	|singlebandPseudocolor|		|multibandColor|
	|pointCloudExtent|		
	|sum|		|sort|

continues on next page

Table 5.4 – continued from previous page

Icon	Substitution	Icon	Substitution
	|paintEffects|		|mapIdentification|
	|styleManager|		|iconView|
	|joinNotEditable|		|joinedLayerNotEditable|
	|joinHasNotUpsertOnEdit|		|filterTableFields|
	|symbologyEdit|		
	|sharingImport|		|sharingExport|

5.9 Plugins

5.9.1 Processing

Icon	Substitution	Icon	Substitution
	|processingAlgorithm|		|processingModel|
	|processingHistory|		|processingResult|
	|menu|		|runSelected|
	|processSelected|		|editHelpContent|
	|saveAsPython|		|modelOutput|
	|qgsProjectFile|		|addToProject|
	|fieldInteger|		
	|meanCoordinates|		|extractLayerExtent|
	|selectRandom|		|vectorGrid|
	|convexHull|		|buffer|
	|intersect|		|union|
	|symmetricalDifference|		|clip|
	|difference|		|dissolve|
	|checkGeometry|		|exportGeometry|
	|delaunay|		|centroids|
	|polygonToLine|		|extractVertices|
	|lineToPolygon|		|nearestNeighbour|
	|splitLayer|		|heatmap|
	|showRasterCalculator|		|showMeshCalculator|
	|regularPoints|		|addGeometryAttributes|
	|basicStatistics|		|uniqueValues|
	|collect|		|simplify_2|
	|createGrid|		|distanceMatrix|

continues on next page

Table 5.5 – continued from previous page

Icon	Substitution	Icon	Substitution
	|lineIntersections|		|mergeLayers|
	|sumPoints|		|sumLengthLines|
	|randomPointsInPolygons|		|randomPointsWithinPolygon|
	|randomPointsOnLines|		|randomPointsWithinExtent|
	|multiToSingle|		
	|grid|		|tiles|
	|merge|		|rasterClip|
	|contour|		|proximity|
	|polygonize|		|rasterize|
	|sieve|		|nearblack|
	|projectionAdd|		|projectionExport
	|8To24Bits|		|24To8Bits|
	|rasterInfo|		|rasterOverview|
	|vrt|		|voronoi|
	|translate|		|warp|
	|iterate|		|terminal|
	|queryHistory|		|storedqueries|

5.9.2 Various Core Plugins

Standard provided with basic install, but not loaded with initial install

Icon	Substitution	Icon	Substitution
	|showPluginManager|		|installPluginFromZip|
	|pythonFile|		|runConsole|
	|showEditorConsole|		|clearConsole|
	|scriptOpen|		|searchEditorConsole|
	|searchRegex|		|replace|
	|commentEditorConsole|		|formatCode|
	|classBrowserConsole|		|codepadConsole|
	|syntaxErrorConsole|		
	|offlineEditingCopy|		|offlineEditingSync|
	|plugin|		|metasearch|
	|geometryChecker|		|topologyChecker|
	|fromSelectedFeature|		|sqlQueryBuilder|

5.9.3 Grass integration

Icon	Substitution	Icon	Substitution
	`|grassLogo|`		`|grassRegion|`
	`|grassTools|`		`|grassNewMapset|`
	`|grassOpenMapset|`		`|grassCloseMapset|`