translator – Easy translation of strings in LaTeX*

Joseph Wright†

Released 2020-08-03

Contents

1 Translating Strings

1.1 Introduction

The translator package is a LaTeX package that provides a flexible mechanism for translating individual words into different languages. For example, it can be used to translate a word like "figure" into, say, the German word "Abbildung". Such a translation mechanism is useful when the author of some package would like to localize the package such that texts are correctly translated into the language preferred by the user. The translator package is *not* intended to be used to automatically translate more than a few words.

*This file describes v1.12c, last revised 2020-08-03.

†E-mail: joseph.wright@morningstar2.co.uk

You may wonder whether the translator package is really necessary since there is the (very nice) babel package available for LaTeX. This package already provides translations for words like "figure". Unfortunately, the architecture of the babel package was designed in such a way that there is no way of adding translations of new words to the (very short) list of translations directly built into babel.

The translator package was specifically designed to allow an easy extension of the vocabulary. It is both possible to add new words that should be translated and translations of these words.

The translator package can be used together with babel. In this case, babel is used for language-specific things like special quotation marks and input short-cuts, while translator is used for the translation of words.

1.2 Basic Concepts

1.2.1 Keys

The main purpose of the translator package is to provide translations for *keys*. Typically, a key is an English word like `Figure` and the German translation for this key is "Abbildung".

For a concept like "figures" a single key typically is not enough:

1. It is sometimes necessary to translate a group of words like "Table of figures" as a whole. While these are three words in English,the German translation in just a single word: "Abbildungsverzeichnis".

2. Uppercase and lowercase letters may cause problems. Suppose we provide a translation for the key `Figure`. Then what happens when we want to use this word in normal text, spelled with a lowercase first letter? We could use TeX's functions to turn the translation into lowercase, but that would be wrong with the German translation "Abbildung", which is always spelled with a capital letter.

3. Plurals may also cause problems. If we know the translation for "Figure", that does not mean that we know the translation for "Figures" (which is "Abbildungen" in German).

Because of these problems, there are many keys for the single concept of "figures": `Figure`, `figure`, `Figures`, and `figures`. The first key is used for translations of "figure" when used in a headline in singular. The last key is used for translations of "figure" when used in normal text in plural.

A key may contain spaces, so `Table of figures` is a permissible key.

Keys are normally English texts whose English translation is the same as the key, but this need not be the case. Theoretically, a key could be anything. However, since the key is used as a last fall-back when no translation whatsoever is available, a key should be readable by itself.

1.2.2 Language Names

The translator package uses names for languages that are different from the names used by other packages like babel. The reason for this is that the names used by babel are a bit of a mess, so Till decided to clean things up for the translator package. However, mappings from babel names to translator names are provided.

The names used by the translator package are the English names commonly used for these languages. Thus, the name for the English language is `English`, the name for German is `German`.

Variants of a language get their own name: The British version of English is called `BritishEnglish`, the US-version is called `AmericanEnglish`.

For German there is the special problem of pre-1998 as opposed to the current (not yet fixed) spelling. The language `German` reflects the current official spelling, but `German1997` refers to the spelling used in 1997.

1.2.3 Language Paths

When you request a translation for a key, the translator package will try to provide the translation for the current *language*. Examples of languages are German or English.

When the translator looks up the translation for the given key in the current language, it may fail to find a translation. In this case, the translator will try a fall-back strategy: it keeps track of a *language path* and successively tries to find translations for each language on this path.

Language paths are not only useful for fall-backs. They are also used for situations where a language is a variant of another language. For example, when the translator looks for the translation for a key in Austrian, the language path starts with Austrian, followed by German. Then, a dictionary for Austrian only needs to provide translations for those keys where Austrian differs from German.

1.2.4 Dictionaries

The translations of keys are typically provided by *dictionaries*. A dictionary contains the translations of a specific set of keys into a specific language. For example, a dictionary might contain the translations of the names of months into the language German. Another dictionary might contain the translations of the numbers into French.

1.3 Usage

1.3.1 Basic Usage

Here is a typical example of how to use the package:

```
\documentclass[german]{article}
\usepackage{babel}
\usepackage{some-package-that-uses-translator}
\begin{document}
```

```
...
\end{document}
```

As can be seen, things really happen behind the scenes, so, typically, you do not really need to do anything. It is the job of other package to load the translator package, to load the dictionaries and to request translations of keys.

1.3.2 Providing Translations

There are several commands to tell the translator package what the translation of a given key is. As said before, as a normal author you typically need not provide such translations explicitly, they are loaded automatically. However, there are two situations in which you need to provide translations:

1. You do not like the existing translation and you would like to provide a new one.
2. You are writing a dictionary.

`\newtranslation` `\newtranslation[`⟨*options*⟩`]{`⟨*key*⟩`}{`⟨*translation*⟩`}`

This command defines the translation of ⟨*key*⟩ to be ⟨*translation*⟩ in the language specified by the ⟨*options*⟩.

You can only use this command if the translation is really "new" in the sense that no translation for the keys has yet been given for the language. If there is already a translation, an error message will be printed.

The following ⟨*options*⟩ may be given:

- `to =` ⟨*language*⟩

  This options tells the translator that the ⟨*translation*⟩ of ⟨*keys*⟩ applies to the ⟨*language*⟩.

  Inside a dictionary file (see Section 1.3.3), this option is set automatically to the language of the dictionary.

For example, one might use

```
\newtranslation[to = German]{figure}{Abbildung}
\newtranslation[to = German]{Figures}{Abbildungen}
```

in a document to define these translations where they are not already available.

`\renewtranslation` `\renewtranslation[`⟨*options*⟩`]{`⟨*key*⟩`}{`⟨*translation*⟩`}`

This command works like `\newtranslation`, only it will redefine an existing translation.

`\providetranslation` `\providetranslation[`⟨*options*⟩`]{`⟨*key*⟩`}{`⟨*translation*⟩`}`

This command works like `\newtranslation`, but no error message will be printed if the translation already exists. It this case, the existing translation is not changed.

This command should be used by dictionary authors since their translations should not overrule any translations given by document authors or other dictionary authors.

\deftranslation `\deftranslation[⟨options⟩]{⟨key⟩}{⟨translation⟩}`

This command defines the translation "no matter what". An existing translation will be overwritten.

This command should typically used by document authors to install their preferred translations.

Here is an example where a translation is provided by a document author:

```
\documentclass[ngerman]{article}
\usepackage{babel}
\usepackage{some-package-that-uses-translator}
\deftranslation[to=German]{Sketch of proof}{Beweisskizze}
\begin{document}
...
\end{document}
```

1.3.3 Creating and Using Dictionaries

Two kind of people will create *dictionaries*: First, package authors will create dictionaries containing translations for the (new) keys used in the package. Second, document authors can create their own private dictionaries that overrule settings from other dictionaries or that provide missing translations.

There is not only one dictionary per language. Rather, many different dictionaries may be used by translator when it tries to find a translation. This makes it easy to add new translations: instead of having to change translator's main dictionaries (which involves, among other things, the release of a new version of the translator package), package authors can just add a new dictionary containing just the keys needed for the package.

Dictionaries are named according to the following rule: the name of the dictionary must start with its *kind*. The kind tells translator which kind of keys the dictionary contains. For example, the dictionaries of the kind `translator-months-dictionary` contain keys like `January` (note that this is a key, not a translation). Following the kind, the name of a dictionary must have a dash. Then comes the language for which the dictionary file provides translations. Finally, the file name must end with `.dict`.

To continue the example of the month dictionary, for the German language the dictionary is called `translator-months-dictionary-German.dict` Its contents is the following:

```
\ProvidesDictionary{translator-months-dictionary}{German}

\providetranslation{January}{Januar} \providetranslation{February}{Februar}
\providetranslation{March}{M\"arz} \providetranslation{April}{April}
\providetranslation{May}{Mai} \providetranslation{June}{Juni}
\providetranslation{July}{Juli} \providetranslation{August}{August}
\providetranslation{September}{September} \providetranslation{October}{Oktober}
\providetranslation{November}{November} \providetranslation{December}{Dezember}
```

Note that the `\providetranslation` command does not need the option `[to = German]`. Inside a dictionary file translator will always set the default translation language

to the language provided by the dictionary. However, you can still specify the language, if you prefer.

\ProvidesDictionary `\ProvidesDictionary{⟨kind⟩}{⟨language⟩}[⟨version⟩]`

This command currently only prints a message in the `.log` file. The format is the same as for LaTeX's `\ProvidesPackage` command.

Dictionaries are stored in a decentralized manner: a special dictionary for a package will typically be stored somewhere in the vicinity of the package. For this reasons, translator needs to be told which *kinds* of dictionaries should be loaded and which *languages* should be used. This is accomplished using the following two commands.

\usedictionary `\usedictionary{⟨kind⟩}`

This command tells the translator package, that at the beginning of the document it should load all dictionaries of kind ⟨*kind*⟩ for the languages used in the document. Note that the dictionaries are not loaded immediately, but only at the beginning of the document.

If no dictionary of the given *kind* exists for one of the language, nothing bad happens.

Invocations of this command accumulate, that is, you can call it multiple times for different dictionaries.

\uselanguage `\uselanguage{⟨languages⟩}`

This command tells the translator package that it should load the dictionaries for all of the ⟨*languages*⟩. The dictionaries are loaded at the beginning of the document.

Here is an example of how all of this works. Suppose you wish to create a new package for drawing, say, chess boards. Let us call this package chess. In the file `chess.sty` we could now write the following:

```
\RequirePackage{translator}
\usedictionary{chess}
...

\newcommand\MoveKnight[2]{%
  ...
  \translate{knight}
  ...
}
```

Now we create dictionaries `chess-German.dict`

```
\ProvidesDictionary{chess}{German}

\providetranslation{chess}{Schach}
\providetranslation{knight}{Springer}
\providetranslation{bishop}{L\"aufer}
...
```

and `chess-English.dict`

```
\ProvidesDictionary{chess}{English}
```

```
\providetranslation{chess}{chass}
\providetranslation{knight}{knight}
\providetranslation{bishop}{bishop}
...
```

Here are a few things to note:

- The package `chess.sty` does not use the command `\uselanguage`. After all, the package does not know (or care) about the language used in the final document. It only needs to tell the translator package that it will use the dictionary `chess`.
- You may wonder why we need an English dictionary. After all, the keys themselves are the ultimate fall-backs if no other translation is available. The answer to this question is that, first of all, English should be treated like any other language. Second, there are some situations in which there is a "better" English translation than the key itself. An example is explained next.
- The keys we chose may not be optimal. What happens, if some other package, perhaps on medieval architecture, also needs translations of knights and bishops. However, in this different context, the translations of knight and bishop are totally different, namely `Ritter` and `Bischof`.

  Thus, it might be a good idea to add something to the key to make it clear that the "chess bishop" is meant:

  ```
  \providetranslation{knight (chess)}{Springer}
  \providetranslation{bishop (chess)}{L\"aufer}
  ```

  for German and

  ```
  \providetranslation{knight (chess)}{knight}
  \providetranslation{bishop (chess)}{bishop}
  ```

  for English.

1.3.4 Creating a User Dictionary

There are two ways of creating a personal set of translations. First, you can simply add commands like

```
\deftranslation[to = German]{figure}{Figur}
```

to your personal macro files.

Second, you can create a personal dictionary file as follows: In your document you say

```
\documentclass[ngerman]{article}
\usepackage{translator}
\usedictionary{my-personal-dictionary}
```

and then you create the following file somewhere where TEX can find it:

```
\ProvidesDictionary{my-personal-dictionary}{German}

\deftranslation{figure}{Figur}
```

1.3.5 Translating Keys

Once the dictionaries and languages have been setup, you can translate keys using the following commands.

`\translate` `\translate[`⟨*options*⟩`]{`⟨*key*⟩`}`

This command will insert the translation of the ⟨*key*⟩ at the current position into the text. The command is robust.

The translation process of ⟨*key*⟩ works as follows: translator iterates over all languages on the current *language path* (see Section 1.3.6). For each language on the path, translator checks whether a translation is available for the ⟨*key*⟩. For the first language for which this is the case, the translation is used. If there is no translation available for any language on the path, the ⟨*key*⟩ itself is used as the translation.

The following options may be given:

- `to =` ⟨*language*⟩

  This option overrules the language path setting and installs ⟨*language*⟩ as the target language(s) for which translator tries to find a translation.

`\translatelet` `\translatelet[`⟨*options*⟩`]{`⟨*macro*⟩`}{`⟨*key*⟩`}`

This command works like the `\translate` command, only it will not insert the translation into the text, but will set the macro ⟨*macro*⟩ to the translation found by the `\translate` command.

1.3.6 Language Path and Language Substitution

`\languagepath` `\languagepath{`⟨*language path*⟩`}`

This command sets the language path that is searched when translator looks for a key.

The default value of the language path is `\languagename,English`. The `\languagename` is the standard TEX macro that expands to the current language. Typically, this is exactly what you want and there is no real need to change this default language path.

There is a problem with the names used in the macro `\languagename`. These names, like `ngerman`, are not the ones used by translator and we somehow have to tell the translator about aliases for cryptic language names like `ngerman`. This is done using the following command:

`\languagealias` `\languagealias{`⟨*name*⟩`}{`⟨*language list*⟩`}`

This command tells the translator that the language ⟨*name*⟩ should be replaced by the language in the ⟨*language list*⟩, for example

```
\languagealias{ngerman}{German}
\languagealias{german}{German1997,German}
```

For the languages used by the babel package, the aliases are automatically set up, so you typically do not need to call either `\languagepath` or `\languagealias`.

1.3.7 Package Loading Process

The translator package is loaded "in stages":

1. First, some package or the document author requests the translator package is loaded.
2. The translator package allows options like `ngerman` to be given. These options cause the necessary aliases and the correct translator languages to be requested.
3. During the preamble, packages and the document author request creating dictionary kinds and certain languages to be used. There requests are stored for loading later.
4. At the beginning of the document the requested dictionary–language pairs are loaded.

The first thing that needs to be done is to load the package. Typically, this is done automatically by some other package, but you may wish to include it directly:

When you load the package, you can specify (multiple) babel languages as ⟨*options*⟩. The effect of giving such an option is the following: It causes the translator package to call `\uselanguage` for the appropriate translation of the babel language names to translator's language names. It also causes `\languagealias` to be called for the languages.

2 Contributing

Since this package is about internationalization, it needs input from people who can contribute translations to their native tongue.

In order to submit dictionaries, please do the following:

1. Read this manual and make sure you understand the basic concepts.
2. Find out whether the translations should be part of the translator package or part of another package. In general, submit translations and new keys to the translator project only if they are of public interest.

   For example, translations for keys like `figure` should be send to the translator project. Translations for keys that are part of a special package should be send to the author of the package.

3. If you are sure that the translations should go to the translator package, create a dictionary of the correct name (see this documentation once more).

4. Finally, submit the dictionary on the development site: https://github.com/josephwright/translator.