Some of the content of this book is based on material from the website of the Translate project:

http://translate.sourceforge.net/

Project conception and coordination	Manal Hassan (منال حسن) Dwayne Bailey
Author	Friedel Wolff
Additional content	Samuel Murray – terminology Khaled Hosny (خالد حسني) – transliteration
Proofreading and feedback	Hermien Bos Kenneth Nielsen
Illustration and covers	Heather Bailey
French translation *La localisation au service d'un changement durable*	Mohomodou Houssouba Claude Paroz
Arabic translation إحداث التغيير بتوطين المعلوماتية	Khaled Hosny (خالد حسني) Ahmad Gharbeia (أحمد غربية)
Spanish translation *La localización al servicio de un cambio verdadero*	Lucía Morado Vázquez Silvia Rodríguez Vázquez
Funding	International Development Research Centre (IDRC)

Introduction

Computers are amazing tools. They can help us in so many ways, yet we are often frustrated when they don't work the way we want them to. We are frequently in interaction with technology where we change the way they work, and they change the way we work. A lot of this change is good, but we are not always aware how it changes us.

A lot of technologies we work with often, such as software, web sites and cell phones, contain text which is in some language or another. Today, it is often in one of the major languages of the world, such as English or French. Those of us who understand the language of the technology are able to use it, while other people might be entirely locked out, or unable to fully enjoy the benefits that the technology brings.

This book explains how software can be *localised* – the process whereby software is translated or adapted in some other way. It will empower the user with knowledge of the issues and tools involved in translating software into new languages. Translated software can ensure that technology is usable for more people, and to ensure that more languages become used in information technology.

It is important to realise that even if we are able to use technology in our second or third language, the constant exposure to this language doesn't only allow us to use the technology, but it also slowly degrades our ability to use our own language. It is my belief that languages that are not used in all the important spheres of our lives will have a hard time surviving the onslaught of globalisation. Software localisation can therefore not only help with access to information and useful technology, but also with language revival and cultural preservation.

This book specifically discusses the localisation of so-called *Free and Open Source Software* (FOSS) – software that we are able to localise and distribute freely. Many such software projects are leading the industry in language support. Although this book is about FOSS, it will also be useful for learning about software localisation in general, and about contributing to FOSS projects in more ways than just localisation.

While several topics are discussed in this book, the scope is obviously limited to keep it at a manageable size. Many more topics might be relevant to localisation in a specific language, or with certain tools. We limit the discussion to topics of general interest to most FOSS localisation, and we limit the discussion on tools mostly to Virtaal and Pootle.

Content

Chapter 1: Background

Software localisation

Computer software is often created to communicate with the user in a certain language, such as English. For the software to be usable by people who speak a different language, the software messages would have to be translated. This process is at the heart of *software localisation*. Localisation really refers to all the changes made to a product to adapt it to a new language, country, culture or even legislation.

Example

Look at how this text editor is completely adapted for Arabic:

Although translation is the most prominent part of software localisation, there are many other changes to software that might be considered to adapt it to a new set of users. Maybe some images should be replaced, references to money should be adapted to the

local currency, and dates and numbers might be formatted in a different way depending on the writing tradition of the new users.

Software companies often consider software localisation when they are trying to find new clients in a country they didn't operate in before, and localisation is a huge industry. Whereas even the most popular software was only available in a few languages during the 1990s, it is common these days for some software to be available in more than 30 languages, with some even available in more than 50.[1]

Making software available in a new language allows a whole new group of people to use software who might not have been able to do so before. Localised software might be a crucial requirement for someone to be able to take part in the digital world. As we spend more time with computers, and more people do their jobs with computers, the need increases for everyone to be able to use computers, even if they only know one language well. By localising software, you can empower users to be first-class citizens of the digital world, without being slowed down by a language barrier or needing to feel as if their language is unimportant. Using your language in the realm of information technology is part of keeping it alive and relevant to the everyday activities of people.

Open Source localisation

Although lots of companies sell computer software or only give you a licence to use their software under many restrictions, lots of software is available without such restrictions and often at no cost. Such software is often called *Free Software* or *Open Source Software*. Great emphasis is placed on the *freedoms* given to users of the software, and to the availability of the software for any purpose. The localisation of this type of software is the focus of this book.

Read more about Free Software: http://www.gnu.org/philosophy/free-sw.html
Read more about Open Source Software: http://opensource.org/docs/osd
Free and Open Software is often abbreviated as FOSS – an abbreviation that is also used in this book. Software released under stricter licences which is not Free and Open Source Software, is often called *proprietary software*. Although some proprietary software is aimed at the global market and available in many languages, it is usually entirely up to

1 Mozilla Firefox version 3.6 was released in 65 localised versions, with more languages available as extensions.

the software developer to decide which languages to support, and whether or not to release the localised versions at the same time as the official version.

Because Free Software allows modification, anyone is allowed to adapt the software to their needs. If we believe software localisation is important, Free Software allows us to localise the software and to distribute the modified version. In reality, Free Software is usually created by a community with an interest in seeing the software succeed in many ways, and welcome localisers to take part in the project to improve the software and contribute translations and other improvements necessary to make it more suited for a wider audience.

Since FOSS is valuable because it allows us to do more, and because it is usually available at little or no cost, we might want to localise FOSS simply as a way to support these projects, and to make them more popular.

Therefore we see many reasons for localising FOSS:

- Anyone is free to contribute and cause the change they want to see for their language
- You can promote software that values your freedom
- You can influence developers of proprietary software by improving their competition
- You can gain experience in localisation and translation

Differences from normal translation

Any professional translator will tell you that translation isn't just some mechanical process. It is a creative process, a topic of academic research, and students can study translation at many institutions. When we see how software struggles to translate from one language to another, we see that a translation requires attention to detail to really feel right. The principles of document translation apply to software localisation as well, and you will definitely benefit from any knowledge of translation. Don't think that any computer programmer can translate software well just because they understand computers.

However, there are several areas in which software translation is different from other types of translation. Although the text is often of a more technical nature, that is not the only issue. Different tools are used, and there are different constraints on what you can do in a translation. We devote a whole chapter in this book to some of the technical issues that appear in software localisation (see page 35). Therefore, keep in mind that

very skilled translators will still need to be trained in the issues particular to software translation.

An interesting thing to keep in mind is that a translated document is usually read in a predictable way, from top to bottom, and seen in the same way it left the hands of the translator. With software, it is impossible to know exactly what a user will see, and in which order. Some of the text is changed when the software is running. A user might use software differently from you and therefore experience your translation in a different way than you do when using the translated software!

Differences from commercial localisation

If you are familiar with commercial localisation, you probably already saw that FOSS localisation can be quite different. Whether or not money is involved, can obviously make a big difference to the motivations that participants have. It means that a translator might have much more freedom. Workflows are usually not dictated by the software project.

There is often much closer contact between programmers and localisers in FOSS projects, and therefore you can usually get quick feedback on any questions or problems with the original text. Users and translators are also often in more direct contact, allowing feedback to reach you more directly.

Many FOSS projects are really welcoming to localisers, and realise that they need to be attractive to volunteers, and often work hard to make things easy for volunteers to contribute translations. For this reason, it often happens that popular FOSS projects get contributions in many languages, even for languages not yet commercially viable outside the FOSS world.

In terms of tools and file formats, obviously FOSS translation tools are the preferred tools to use for FOSS localisation. The PO format is by far the most popular file format. Since this format contains both original and translation (similar to XLIFF), there is a lesser focus on retranslation and the exchange of translation memories. An example workflow is discussed in the section on “A simple localisation process” on page 59.

Chapter 2: Orientation

What do you want to achieve?

Formulating objectives

As we saw in the previous section, there are many reasons to localise FOSS. Your reasons might be different from other contributors, but that usually won't be a problem. What you should always keep in mind is this question:

What do you want to achieve?

If you just want to localise software because it is fun, it probably doesn't matter much which software you localise or how you do it. However, many people might have specific goals. Maybe your main goal is to empower people of your country. Maybe you have language preservation in mind. Maybe you just want to promote some software. All of these are valid and noble goals, but it is useful to articulate them and keep your goals in mind, since they can influence your decisions.

Whether you work on your own or in a group or even a formal organisation, clear goals will help to motivate and to keep your focus on what you want to achieve. Many companies work towards formulating a vision and mission as part of their strategic planning. Knowing what you are trying to achieve not only helps you to keep an eye on the target, but also makes it easier to know when you succeeded!

If you are working as a volunteer, you should try to make your localisation work as much fun as possible. Set it as a secondary goal if you want.

Setting goals to reach objectives

At Translate.org.za in South Africa we wanted to empower people and uplift their languages. At least, that is what we told ourselves, but initially we were not working along the best path to achieve that goal – we were translating software that we liked, instead of software that people are actually likely to use. Since we couldn't possibly translate all software, we had to choose, which means we had to prioritise. If you

understand your goals better, you will find more constructive ways to prioritise different projects, and hopefully, reach your goals faster or more successfully.

Since you will try to reach your goals with limited resources (volunteers, money, time, etc.) you should plan for continuity and sustainability. Think of how you will motivate volunteers. If money is involved, set up a budget and see how well you keep to it. Measure the work so that you become good at time planning. We will discuss some techniques for evaluating projects and prioritising your work in the section on "Prioritisation (when you can't do everything)" on page 73.

Some principles to keep in mind

There are many things to learn about localisation and translation, and it might seem like a big undertaking. However, you can do good localisation while avoiding some important mistakes, and have fun while learning. Here are a few principles to keep in mind.

Start small. Don't think that you need to start with the biggest and most popular pieces of software. Choose a task that you can complete easily, and fits in your available time. You will get a lot of reward and motivation from completing something small.

Read the instructions. You can often find a lot of advice to help you in a given situation. The specific software you are translating might have instructions to help with the difficult parts. Maybe a certain sentence has a comment from the programmer to explain a difficult concept. A language guide for your language might clear up some difficulty you have with spelling or grammar.

Stay consistent. Consistency is an important goal in localisation. You want to make sure that you use terminology consistently, as well as using a consistent style in an application. Furthermore you should aim to harmonise your terminology and style with other applications on the same platform that the application will run on (Windows, KDE, GNOME, OS X, etc.). Then you should aim for consistency with the language use outside cyberspace – what people use at universities, newspapers, magazines, etc. Eventually you will develop your own style, and you don't need to follow other people's mistakes blindly, but don't do things differently without a reason.

Be careful, but be bold. This advice might sound like a contradiction, but try to balance these two opposing goals. You should be careful in what you do, and strive for good quality. Don't be lazy to do your research, and ensure that you or someone else reviews

your translations. Maybe you should leave a translation that you can't do well yet until you have the required information. However, sometimes it might just help your progress if you go ahead and coin the new term, or change a bad habit that has formed in your team. In FOSS you can usually fix your mistakes quite easily, and if you learn from your mistakes, you might gain a lot from not being too careful. Find a balance between being careful and being bold.

Expect criticism. Unfortunately, not everyone will always appreciate your hard work. If you are breaking new ground you stand the chance of attracting criticism. This is to be expected, and isn't necessarily bad. Criticism means that someone paid attention to what you did. Constructive criticism might help to improve your work, to fine-tune your terminology, or to reduce your errors. Especially if you are doing localisation that is entirely new for your language, take into consideration that users sometimes need a bit of time to get used to new terms in a localised user interface. Your audience will develop with you, and new terminology might become established and accepted even if they were criticised initially. Don't ignore criticism, but realise that new localisations usually look surprising and might attract comments.

Enjoy it. Don't get so caught up in the work that you don't enjoy it. Most FOSS localisation is done by volunteers for fun. If the work isn't rewarding in some way, you might not be able to sustain your work, and you might not reach your goals. Translate software that you use and enjoy. Give compliments to the programmers. If you like to get credit for your work, ensure that you work hard on publicity and marketing for your work.

Your environment

The localisation work you do will take place in a certain environment. Not all advice is equally suitable for all situations. If you are aware of your resources, your users and the world they live in, it will help you do better localisation, and work more effectively.

Target audience: speakers of your language

In all translation work it is important to know who you are translating for. Who will read the text that you are writing? How old are they? What level of literacy is expected? If your target audience is young children from a rural school, you will probably translate differently than for university professors! If you are localising into your language,

remember that the speakers of that language are also your primary audience for marketing and promoting your work. Don't unnecessarily use another language like English to support users or discuss terminology.

Target audience: users of open source software

Users of Open Source Software tend to have a stronger community orientation, and like to stay up to date with software news. You might also be able to find volunteers to help with terminology, reviewing, or testing of the translated software. Someone might be willing to help with promotion. People might be more willing to volunteer their time because you are providing localised software that respects the user's freedom.

Keep in mind that while it might be easy to contribute by complaining, few people will really help with substantial contributions. Realise that users might have high expectations, and you might not impress them with bad work.

Your localisation team

If you have big goals you might need lots of help to get there. Building a strong team will help you do more work, and improve the work by bringing in different skills. Always try to contact previous translators that might have done work you are building on. Document what you do, and make sure that people know about it. Think about ways to promote the work you do amongst the people you are trying to reach. Think about newspapers, radio, television, blogging, and social networks.

Try to train newcomers so that they can become useful contributors. Create awareness of localisation. At Translate.org.za, we have often held events called *Translate@thons* where we get people together for a day or two to translate a piece of software. It is a great way to show people what software localisation is, train people in the basics, while having fun as a group and working towards a common goal.

Read more about hosting a Translate@thon:

http://translate.sourceforge.net/wiki/guide/translateathon

If you are working alone, don't let all the references to localisation teams upset you. A lot of FOSS localisation (especially for smaller languages) is done by single volunteers working mostly alone. If localisation is new in your language, keep in mind that someone needs to be first – it could just as well be you. Some people join an existing project more easily than creating a new one.

The role of English

When talking about the environment in which you do software localisation, it is important to say something about the special role of English. English has become a dominant language in the world of software, and this affects the way we do our work. Most software is translated from English into other languages. Some of your users and fellow translators might be very used to English terminology and style and might prefer that.

In FOSS projects, you will find that a lot of communication amongst participants usually takes place in English. If you are able to work comfortably in English, you will take part more easily in such projects, and should also be able to interpret the English texts more easily. When communicating in English with other people, try to clearly explain your points, and review your text to avoid misunderstandings due to spelling or other mistakes.

However, there are many ways to help someone who doesn't know English well. Some software (like Pootle) can show you another language (like French or Arabic) while translating to help you understand the text better. Machine translation systems on the web might help you to communicate with other team members in a language you don't understand well. People who don't know a second language can still be valuable contributors – even if just for marketing and testing.

Orientate yourself

When you are ready to start to translate a specific piece of software, ensure that you are familiar with it first. In the FOSS world, you can usually get the software and try it out without too much trouble. Visit the project website, read through some of the documentation, and try to run the software on your own computer. It will provide you with useful background information on what the software can do, and what it is that you will be translating. Another useful source of information is the help files that might be available inside the application.

If you are starting a new translation for the software in your language, even an older version of the software should give you a very good idea of how the software works, and what the terms being used really mean. Always try to approach your translation with good knowledge of the software rather than relying on the text only.

Chapter 3: Before you start

The tools that we'll use

There are many tools that you could use for translation. It would be impossible to discuss them all here. We will provide most of the explanations with reference to Pootle and Virtaal. They have been developed to help localisers work more productively, manage their teams better, and increase quality.

Pootle is a popular web-based system for translation and translation management. It is used by several FOSS projects to manage their localisation work. Pootle allows online translation, and it has several features to help with review and team organisation. It is ideal for team work, casual contributions, and translation sprints (Translate@thons). Visit the official Pootle server to see what it is all about:

http://pootle.locamotion.org/

While online translation is useful, we realise that internet connectivity is still a problem for many people. **Virtaal** is a program for translating on your computer, even if you don't have internet access. It provides many powerful features to improve productivity without sacrificing quality. Installation instructions and more information are available on the Virtaal website:

http://virtaal.org/

You don't need to choose between these two tools – they complement each other. Even if a project is hosted on Pootle, you can still translate it offline in Virtaal and upload your files later. These tools are described in more detail in the section on "Localisation tools" on page 51. Several sections will refer to specific features of the tools, so it is a good idea for you to have Virtaal installed, and preferably also an account on a Pootle server so you can follow the instructions in the book. Virtaal will provide the most help during your translation with colour highlighting of important things, and integration with helpful resources if you have an internet connection.

Both these programs are built on top of the **Translate Toolkit** – a powerful collection of small utility programs for localisers. The programs in the Translate Toolkit are run from a

command-line window, and therefore won't be discussed much in the book. Installation instructions and more information is available on the website of the Translate Toolkit:

http://translate.sourceforge.net/wiki/toolkit/index

Viewing your language

Before you can do any work, you need to ensure that you can view your language on screen. For many languages this is not a problem any more, but you need to know that you have the necessary fonts installed on your computer. Maybe there is a version of Wikipedia with your language, or a local online newspaper, and you can verify that you can view all the characters correctly. If you are having any problems, these sites might provide some links to fonts and instructions to be able to view your language on screen. It is important to know that you must work with so-called Unicode fonts. You can consult the Unicode font guide: http://www.unifont.org/fontguide/

You will probably spend most of your time in a translation program, so paste some text in the translation program to check that you can view the text there as well, and not just in your web browser. Some applications (like Virtaal) allow you to customise the font settings. You might want to adjust settings for the best display while working.

Furthermore, if you don't have the necessary font on your system by default, it could very likely be true for users of your localised software as well. If they need to install an extra font, ensure that there are good instructions available, customised for different operating systems. Keep in mind that users without fonts won't be able to read a website written in your language without that font, unless you publish instructions in a PDF document with the fonts embedded. (This way the users don't need the font on their computers.) Some emerging web technologies allow a website to supply a font, but you should probably not count on that.

In some cases software can be distributed with fonts to ensure that your language can be viewed correctly in the application. The software developers will be able to tell you if it is possible to distribute fonts with their software.

Typing your language

Now that you are able to view your language on screen, it is important to know that you can type the language. In the system configuration you should be able to select the correct keyboard layout or input method for your language. Make sure that you know how to type all the characters of your language that you will need.

If you are not yet used to typing in your language, it might be worthwhile to practice to improve your speed. The faster you can type, the more work you can do, and the more you will enjoy translation.

Codes and locales – initial support for your language

Whenever you contribute the first translation or support for your language to an application, you will need to identify your language accurately to the software developers. When referring to your language, remember to use the English name of your language, and keep in mind that the developers might not know about about your country and relevant dialects. Identifying the language properly is usually done by means of a two or three letter code for your language. For example, the two letter code for English is 'en', and the two letter code for Russian is 'ru'. It is common in FOSS projects to use two letter codes whenever they are available, and to use three letter language codes when no two letter code exists. These codes come from a standard called ISO 639. More information is available on Wikipedia:

http://en.wikipedia.org/wiki/ISO_639

If your language is spoken in multiple countries, it might be customary to do separate localisations for the different variants of the language in the respective countries. In such a case, the code is augmented with a country code, such as 'fr_CA' for French in Canada, or en_GB for English in Great Britain. Note that the country is usually only specified if there is a need to do separate localisation for each country where the language is spoken. Only specify a country if needed by the software project, or if you know that your localisation is part of a set of different localisations for your language.

For some applications to work correctly in your language, your operating system will need a *locale*. A locale provides information to software about a language and some of its customs, such as the proper way to write dates and times in your language, the correct formatting of numbers, etc. The list of available locales usually indicate which languages

are supported on the system. In some cases, some software also maintain their own locale information in order not to rely on the operating system.

For Linux and certain other software that require locales, you can create a locale if it doesn't yet exist. In the case of Linux locales, you really want a locale for the project called "glibc". This page provides more information:

http://translate.sourceforge.net/wiki/guide/locales/glibc

Although not all applications on Linux require a glibc locale, many do, and it is a very good idea to create one if it doesn't exist yet for your language. Hundreds already exist, so most people wouldn't need to worry about this.

Useful extras

With the previously mentioned things in place, you can localise almost any application, but there are several other tools that might help you with quality, or might make you more productive. See if you can find any of the following to help you during translation:

- spell checker – see if you have a working spell checker in your web browser (for online translation) or in Virtaal, depending on where you will do your work
- rules of spelling, orthography or grammar for your language
- bilingual dictionaries, especially related to the field you are translating in
- English dictionary

Even if these are only available online, they might help you when you get stuck.

Chapter 4: Style

Any piece of writing will show some aspects of style. The original language of the software will also have a certain style. It might be characterised by being formal or informal, for being technical or very accessible, and by respecting the rules of spelling and grammar to some degree.

When we translate, we will create a work with a certain style, and should think about what we want to achieve. Should the work be as formal as the original? Can we use loan words, or should we strictly use well-known words from our language? There might be several such questions that could affect how we approach a translation. Think of your goals and your target audience.

It is advisable to form some guidelines for software localisation in your language. Where there is more than one way of doing something, different translators might choose differently, and the inconsistency might reduce the quality. Without guidelines, newcomers to your team might need to make hard choices which can be demotivating and can reduce productivity. This section describes some aspects of style that you might want to consider. You might want to discuss this with other translators and interested people and work on documenting your recommendations as these rules develop in your team.

Also keep in mind that users will probably get used to the style used in software, as long as it is consistent and in line with your cultural conventions.

Orthography

Most languages have standardised some aspects of writing, such as the alphabet, spelling, use of capitalisation and punctuation. These rules form the *orthography* of the language. Some languages have multiple orthographies (like Hausa which can be written with the Latin or Arabic script), or might have newer versions of the same orthography – if spelling or other writing rules change.

You should be certain about the orthography you are following in your localisation work. You will benefit from publications by institutions for language standardisation, or even

good textbooks if you are not very informed about the writing rules of the language. Furthermore, dictionaries will help with issues of spelling, although spelling is not a big problem in all languages. If your language is not well standardised, you might have a lot of freedom to decide how to do things. See your work as part of the task to standardise the writing in the language.

There are some aspects of orthography that might be well standardised, but different from English. When new members of your team translate, they might forget what the true writing tradition is in your language, and therefore it might be useful to write about any problematic issues in your team documentation.

A common mistake when starting with localisation, is to ignore the capitalisation in the original text, and always or never using capital letters. If your language should use capital letters, this can create a bad impression and affect readability of the text. If your language doesn't use capital letters, this mostly doesn't affect you. However, capitalisation can possibly give a hint about the meaning or use of a message in front of you. Capitalisation could indicate proper names (like brands or product names) and a complete lack of capitalisation might indicate that the translation is part of a bigger sentence.

Another common mistake, is to follow the English capitalisation too closely. A writing style called "Title Case" is very common in English software. When something is written in Title Case, most words start with a capital letter. This is not very common in other languages (German and languages with similar orthographies being notable exceptions). Ensure that you follow your language's rules and customs with regards to capitalisation. In most cases, we will write brand names and product names unchanged.

Examples

Here are some examples from Northern Sotho where capital letters are only used for the start of sentences and proper names. The English is followed for capitalisation at the start of each message, but otherwise the title case is avoided for proper writing according to the rules of Northern Sotho.

English	Not good	Recommended
Help	thušo	Thušo
Save As….	Boloka E le...	Boloka e le...
Font Size	Bogolo Bja Fonte	Bogolo bja fonte
Online Help	Thušo ya Inthaneteng	Thušo ya inthaneteng
Select All	Kgetha Tšohle	Kgetha tšohle
Print PDF File	gatiša faele ya pdf	Gatiša faele ya PDF
Print Acrobat File	Gatiša Faele ya Acrobat	Gatiša faele ya Acrobat
no selection	Ga go kgetho	ga go kgetho

Another area where guidelines for your team might help, is to ensure that attention is paid to punctuation and spacing. If the use of punctuation and spacing is similar to English in your language, it might still be useful to specify that translators should pay attention to whether a message ends with a colon (:), question mark (?), exclamation mark (!) or an ellipse (…). Also keep in mind that an ellipse can be represented with three separate dots, or a special ellipse character which might be hard to type. Virtaal makes it easy to insert the special ellipse character by highlighting it as a placeable in the original text (see page 53 for working with placeables).

If your language uses different punctuation from English, it might be good to remind translators of the differences, and to provide examples. Also note the cases that are the same, to avoid confusion. The original punctuation usually indicates something important, but you don't always need to follow it blindly. For example, you might want to avoid the exclamation mark (!) if it is not so commonly used in your language.

Unusual spacing in the original text might be important, such as a space at the start or the end of a message. It might be used to align text, or to implement spacing between different controls in a program, or when parts of a sentence is joined together. This is often a programming mistake which you should report, but you will still encounter this. The use of two spaces between sentences is usually not very important to follow, although it might help to ensure consistency. The quality checks performed by software (such as Pootle) will try to indicate any difference between the original text and your translation.

If your language uses a different alphabet from English, it might be important to decide when and if transliteration should be used. Also ensure that there is agreement on how transliteration is done in the team. Transliteration is discussed in more detail on page 27.

Grammatical style

There are often more than one way to say the same thing. You can be formal, polite, informal, demanding, verbose, and even rude! The appropriate wording to use in your software translation could depend on several things. Ask yourself some of these questions:

- How old are the users of the software?
- In what kind of setting will they be using this? At home or at work?
- Are there any wordings which might be offensive or impolite?

While we obviously want to write in the correct grammar, we additionally want to choose the ways of saying something that won't offend a user, and will feel appropriate for the particular program. Using business language in software for kids might be unsuitable. Being very informal and casual in accounting software might make a user uncomfortable. Asking the user for information in an impolite way might reduce the user's spontaneity and leave a bad impression of the software.

Some things you might want to describe in your team documentation include:

- How formal should the software be? Should the user be addressed as an older or respected person? You might want to stay on the safe side to avoid giving offence.
- How should you translate questions? You will often encounter the word "*please*" in English. What is the appropriate way to word such requests in your language? A direct translation of "*please*" in your language might sound as if the software is desperately pleading for something, while it intended to politely encourage the user to go ahead and do something. Software also often asks for confirmation. A bad translation for "*Are you sure ... ?*" might sound as if the software doubts the user's ability.
- How should you translate commands? In English software, the text on buttons and links are often command verbs. Many languages don't translate these as commands, but use the infinitive form of the verb. Think about what is most

appropriate for your language. Think about if there might be ambiguity about whether the command is addressed to the user or to the computer. If your language clearly distinguishes causative verbs, you might want to ensure fine attention to such detail in these cases.

Be true to the original text

While your writing culture might encourage you to always take a more formal approach than the English software, you might want to vary your level of formality by looking at the original text. The software used for social networking and instant messaging might feel more natural with a little more relaxed tone.

It is important to always try to convey the correct meaning as intended in the original text. While you might feel you can translate something to sound much better than the original text, you should ensure that you are not changing the intended message of the text. Refer to some of the principles on page 25. For example, a translation shouldn't be more specific or more general than the original text. The text might be used in unexpected ways, and a translation too far from the original meaning might be inappropriate for some situations.

Be true to the target language

Earlier we encouraged both **being careful** and **being bold**. This section might similarly seem to contradict the previous section by encouraging conformance to the target language where the previous section encouraged conformance to the original text. These are both important goals, and won't usually conflict with each other. We want to translate so that the result is both well written in the target language, and conveys the same message as the original text.

Your translation should always be true to the style of your language. Don't fall into the trap of using the word order of the English text if it isn't the most natural way to say it in your language. Don't translate expressions literally. Change cultural references where it makes sense. Be aware of taboo words or words with unintended meanings that might distract the user from the message.

The developer documentation of a project might discourage the use of the passive voice. This might only apply to English. Unless this is specifically meant to apply to translations, feel free to ignore this advice if the passive voice is a good style to use in your language.

Chapter 5: Terminology

Terms are words or short phrases with specific meaning. If we want to do good translation, we are obviously interested in terms and their translations. We can consider short phrases under terminology work, partly because they are often defined in similar ways, and we can usually handle them in dictionaries and translation tools in the same way. They also provide an opportunity to capture some aspects of style.

Examples

- file
- download
- network connection
- secure
- Are you sure you want

As you can see, a term can consist of one or several words. They can be nouns, verbs, or any other type of word, or even a short phrase that we might find in translation work.

Why develop terminology?

Developing terminology is an important and ongoing task while translating software. Of course, this is also true of most aspects of translation style. Using good, consistent terminology and style will help users to be comfortable with the software, and will give them pride in using translated software rather than being ashamed of bad quality. Bad translations can also damage the image of the software.

New terms are constantly used in software that will need to be translated. Terminology creation will therefore never finish while you translate software. However, if you have good translations for common terms, it will help all your team members to translate more productively, and will of course help with the consistency. You don't need a dictionary with thousands of terms before you can do useful localisation work.

Translation with lots of difficult terms can be frustrating for new translators, and they might make lots of errors if they are not familiar with the computer terms. Therefore it is important for the motivation of especially new team members to have some existing terms specified, so that they have some help from the start. A good terminology list is a big help in a Translat@thon for motivation and quality.

Another reason why good terminology is important, is to ensure that it is actually correct. Some terms are hard to understand, and might be very specific to the software involved. If a translator misunderstands some English term, it can badly affect quality. You will get a lot of benefit if you continually work on terminology as a separate part of your project. Good terminology can ensure that the translated version of the program isn't misleading the users, or giving the wrong impression. Remember that you don't just want any translation, and in cases where a term should be translated in two different ways depending on the context, you want to capture that as part of your terminology development.

Short phrases might give the opportunity to unify some stylistic aspects of a translation, such as how requests to a user is made (as we see in the examples above, "Are you sure you want").

Example

```
Are you sure you want to download the file over the secure network
connection?
```

In this example sentence, there are some possibilities for making terminology mistakes. Maybe the translation of "secure" isn't standardised, or the translation of "network connection" is not obvious. The sentence starts with a phrase "Are you sure you want to ..." which can probably be translated in different ways, but inconsistent translations might give a bad impression of the software.

Let's indicate which terms might be recognised from our terminology list by underlining them:

```
Are you sure you want to download the file over the secure network
connection?
```

As you can see, it will be quick and easy to translate this with all of these terms available to use. The translation should be consistent with other translations as well as the prescribed terminology and style, and the translator will also be much more productive if

the translation tool makes it easy to insert these terms without actually having to type them. Of course this example is artificial, but you can expect real benefits from even a small list with a few hundred relevant terms.

Which terms should be developed?

If you are motivated to translate software, it might feel very demotivating to think about doing a lot of terminology work. You want to translate software, not write a dictionary, right? (Of course, writing a dictionary is also very useful.) In the previous section we explained why good terminology is useful and important, but obviously you want to prioritise your work to the most frequently used terms, especially at the start of your work.

You may only want to work on a smaller list of terms initially. Therefore it will be useful to know you are working on the most important ones. You might know of some common terms that should perhaps be in your list, but trying to create the list of terms by yourself is not a good use of your time. It is also more useful to work on terms actually occurring in your translation work, rather than some terms that might be useful, but aren't needed for your current work. A good way to tackle this problem is to use a terminology extraction program, such as **poterminology** from the Translate Toolkit (see page 11). The same feature is also available in Pootle (see page 56). Such software analyses the text for frequently occurring terms and produces a list for you to translate. Some of the suggested terms might not be important or meaningful to work on, but see the list of terms as a suggestion to save you time.

One disadvantage of a terminology extraction program, is that it probably has no knowledge of which terms are most used in the very visible parts of the software. For example, you might get some terms used very frequently in error messages. Of course these are very useful to standardise if you will be translating the error messages, and has all the advantages it offers to the team in terms of productivity and motivation. Do what you think is most useful with the time you have available.

Available resources

There might be many resources available to help you with your terminology work. Remember that you probably want to use an existing, known term rather than creating

something new. Therefore it is a good idea to always look at existing sources of terminology, even if just for inspiration. Even looking at terms in languages related to yours could be helpful.

Here are some places that might be useful:

Open-Tran.eu

The website Open-Tran.eu collects many translations from FOSS projects, and provides a way of searching these translations. Although this can never be an authoritative resource for terminology, you will be able to get a good idea of which terms might already be in use in FOSS projects. If your language does not yet have a lot of useful suggestions in Open-Tran.eu, keep in mind that the latest translations from several projects are imported from time to time, and if you are creating new translations, you will help to make this service more useful over time.

The results from Open-Tran.eu indicate from which project a suggestion originates. Keep in mind that you might want to align your current style and terminology with similar projects. For example, if your current translation is part of the GNOME project, you might want to pay particular attention to the suggestions coming from the GNOME project.

http://open-tran.eu/

Microsoft language portal

The Microsoft language portal contains style guides, terminology lists, and existing translations for many languages, and this might be very useful. If the software you are translating will be running on Microsoft Windows, be aware of the terminology used in the Microsoft products. Users might be more used to them.

http://www.microsoft.com/language/

Wikipedia

Wikipedia is an online encyclopaedia. Although it is not primarily intended as a terminology resource, it covers a very wide range of topics in many languages, and you might be able to see how a term is already translated by another translator at Wikipedia. Of course an article at Wikipedia, even if it is not in your language, might help you to properly understand a concept, which in turn will help to properly define and translate it into your language.

http://www.wikipedia.org/

General principles[2]

Technical correctness

Terms must convey the correct meaning. Watch out for a translation that is more general than the original term, or only refers to part of the original concept. Keep in mind that there might be similar but distinct terms used in the original language that might need to be translated differently, whereas your first guess at a translation might lead you to translating them all in the same way. For example, you might want to make sure your translated terms reflect the differences between “photo”, “drawing”, and “diagram”. Think how the terms “file”, “directory”, and “document” are different and how to properly translate these terms.

On the other hand, the original text might unnecessarily have two terms for the same concept. If you are sure that the two concepts are the same, you might want to translate them with the same term. Only do this if you are sure it is safe to do.

On the other hand, a term might have more than one meaning. Ensure that you translate the correct meaning of the term. Some words in English are both nouns and verbs. The term “list” can be a noun (a grouping of things) or a verb (the action of putting things in a list). Ensure that you translate the correct meaning, or that you add both forms to your terminology list.

Always ensure that you really understand the original concept correctly. Use dictionaries and read up on the web to ensure that you know the exact meaning of a term before trying to translate it. Also check your translated terms to ensure that you are not using them incorrectly, or maybe conveying an unintended meaning associated with the term. Ensure the correct spelling and orthography of your term.

Concept recognisability

Remember that you want ordinary users to understand the terms that you use. It might sound impossible to coin technical terms that ordinary users will understand, but keep in mind that terms are often built from plain words, and that users get used to all sorts of terms that might have been strange a few years ago (also in English).

2 Thank you to Samuel Murray for his material that is used in this section. The original Afrikaans article is available here: http://leuce.com/translate/termhelp.html

Existing terms from engineering, mathematics or office administration might help you to create a term that is much more recognisable than something entirely new.

Back-translatability

Ideally you want to use terms that users will be able to "translate back" to the original term. This is particularly useful in cases where users would already be used to the English terms. An added benefit of being able to translate terms back, is that users can guess more easily for which terms to look for on the web or in untranslated product documentation.

For example: in cases where two terms are closely associated in English, such as "log in/log out" or "enable/disable" it is useful if your translated terms can translate back to such an associated term pair. Ideally your translations should form a similar associated pair. Also try to translate a positive action (such as "enable") with a positive action, and a negative action (such as "disable") with a negative action.

Acronyms, abbreviations, brand names, product names

As a localiser you will often see acronyms, abbreviations and brand names in the text. Acronyms and abbreviations are things like "*e.g.*", "*i.e.*", "*SQL*", "*FTP*". Brand names include the names of products, companies and some technologies, such as "*Firefox*", "*OpenType*", "*Google*", "*Windows*" and "*iPhone*".

Since some of these don't form part of normal conversational language, it might seem as if the only solution is to leave them unchanged as they occur in the original text. This will often be the correct decision, but let us be aware of a few of the issues.

Your language might have established translations for abbreviations like "etc.". However, many languages don't have a writing tradition of abbreviation. You might want to simply translate some abbreviations in full form. If the abbreviation is the name of a technology, such as "*XML*" it is often best left unchanged, since the target audience might know what it is, and only know this name. In document translation (such as help files or web sites) you might have space to also add a translated name for reference, but in software you will probably not have enough space to do that.

Some companies and products might have local names in your language, and it might be appropriate to use them. On the other hand, some names may be trade marks, and you might not be allowed to change them. Special exceptions are sometimes made for languages that don't use the Latin script, such as Japanese and Arabic. It is always best to ask in this case.

Some product names are plain words, and you might even be encouraged to translate them. Don't blindly keep the original name of the software if it might be intended to be understandable in your language and readable in your script. Utility applications like the calculator and text editor are often referred to by general words – not names associated with the specific application.

Always take care to follow the capitalisation of the original product name. Using capitalisation in unusual places (or not at all) might be part of the brand identity of the company or product.

Transliteration[3]

If your language doesn't use the Latin script (like Chinese or Arabic), a policy on transliteration will be useful. Here are some guidelines that might help your team:

- If the name is a word explaining what it does, simply translate the name.
- If the name is obscure or very technical in nature, you can consider using it without transliteration if it doesn't occur very often.
- If the name has no obvious meaning, transliterate the pronunciation of the name. If this is too long for an acronym (such as HTML), you might want to transcribe each letter with an equivalent letter in your script, possibly separating them with dots or spaces if it might be confused with a word.

Terminology assistance in translation tools

A good dictionary can be very useful for your work. However, it can be cumbersome to look up words very often. Furthermore, someone might not know when it is important to check in a dictionary and could make incorrect assumptions.

If a translation tool provides terminology assistance, it can be a great help, since the translators can check the prescribed terminology more often, and might be confronted

3 Thank you to Khaled Hosny suggesting guidelines on transliteration. The original Arabic article from the Arabeyes project is available here: http://wiki.arabeyes.org/تعريب_الأسماء

with it even when they didn't consider checking. It should also make it easy to insert a term without typing, which can save time and avoid typing errors.

Pootle can host a terminology project on the server with project specific terminology lists. For more information see page 56.

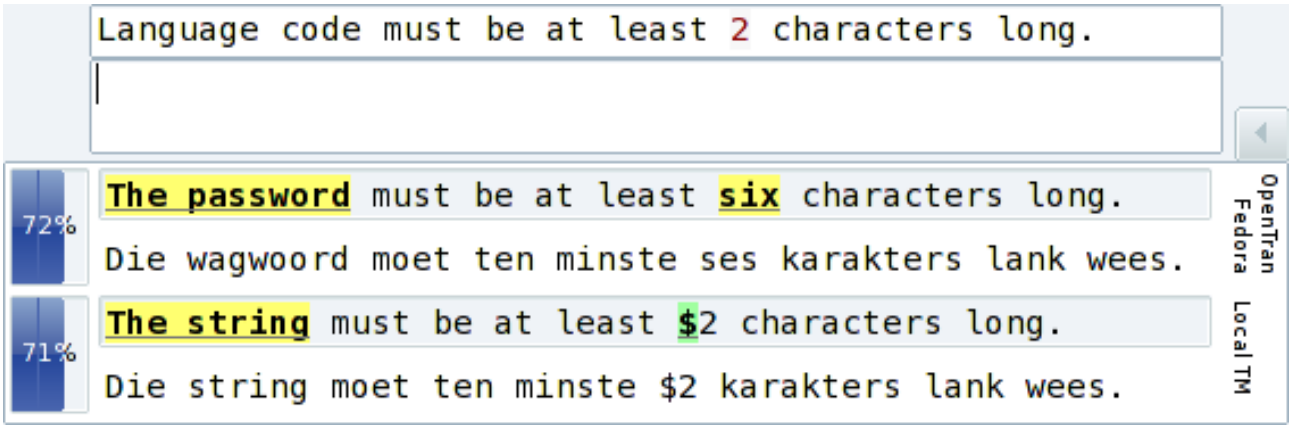
Virtaal can provide terminology suggestions from multiple sources, including terminology files on your computer, and certain network based services. It also provides easy access to look up phrases on websites. For more information see page 53 and the Virtaal preferences.

Chapter 6: Reuse

During translation, a translator creates valuable information – terminology and translations that can be useful later when translating something similar. Being able to reuse previous work is a crucial part of becoming more productive, and ensuring good consistency between different versions of a product, and between different products.

Example

Here you can see how similar translations can be shown to a translator during translation.



Storing and reusing previous translations is called *translation memory*. This is different from terminology in that complete translations or sentences are handled, and software might provide a suggestion even if it isn't quite the same as what you are translating. So while whole words might be wrong in the suggestion, it provides a way to save a lot of typing, and ensures a consistent sentence structure between similar translations.

When should translations be reused?

Any existing translations of sufficient quality should normally be considered for reuse. You might want to give preference to related translations, such as those for the desktop environment you are translating for. For example, you might prefer GNOME translations when translating a GNOME application, rather than KDE translations. In reality the style of translations between different desktop environments won't differ substantially for most

languages, and you will probably benefit from using all available translations in your translation memory.

Short messages are more likely to be ambiguous. Translations consisting of single terms such as “view” and “alert” probably have different translations depending on context. Even longer entries like “Empty trash” is ambiguous (it could describe the state of the trash, or be a command to make the trash empty). Blindly reusing old translations in cases like these are obviously dangerous. All reuse of existing translations should be reviewed to ensure that they are suitable for the current application in terms of style and terminology. Reviewing the suggestions might also provide an opportunity to find errors that might exist in previous translations. Use the opportunity to fix the original source of the error if a suggestion contains a mistake.

Some available sources

The most important source of translation memory, is any existing translations for the project you are working on. You also want to combine any translations for the software, help files, web site, etc. to speed up your work and ensure better consistency.

Next you should look for files of related applications. Files from the same desktop environment (such as GNOME or KDE) is easily available for reuse. This is one of the powerful aspects of FOSS localisation – you can usually easily get existing work.

Next you might want to look for translations of similar programs. If you are translating an e-mail program, look for translations of other e-mail programs to use. If you are translating a graphics editor, maybe translations from a picture viewer will provide some help.

Finally you can also consult existing services where many translations are collected:

- Open-Tran.eu collects translations from several FOSS projects and provides a website to access it. It is also integrated in some translation tools such as Virtaal.

  http://open-tran.eu/

- Microsoft also has a website for their translation memory. Be sure to review the terms of use, and remember that there might be different conventions in the FOSS world in terms of style and terminology.

  http://microsoft.com/language/

Some techniques and tools

The best ways to access existing translations is different for each project, and might change over time. Here are a few useful tips that might save you some time:

GNOME

Existing translations of GNOME are easily available from the website at http://l10n.gnome.org – also known as Damned Lies. Find your language, and select any release set from the list. A link at the bottom of the page provides an archive with all the latest translations.

KDE

Existing translations of KDE is easily available from the website at http://l10n.kde.org/. Find your language team. A link under the section "Archives" provides an archive with several files, including translations in the "messages" directory.

Debian

The Debian project develops several pieces of software, such as the Debian Installer and the ISO codes package (with translations of language and country names). A single file with the translation memory of all projects developed by the Debian project is available here:

http://i18n.debian.net/compendia/

Translations of almost all packages distributed with the Debian project is available here:

http://i18n.debian.net/material/po/

Although it might be better to obtain translations from the specific projects directly, getting them here might be more convenient, while possibly being a bit out of date.

Pootle servers

Pootle is used to manage the translations of many projects such as OpenOffice.org, and you can usually obtain the translations easily from a Pootle server.

Visit the page for your language in a specific project on the server, and go to the "Translate" tab. You should see a link to download a ZIP file with all the files inside. If it isn't visible, you might need to log in, or ask for permission from the site administrator.

Mozilla

The current Mozilla localisation process makes it non-trivial to obtain translations, although they are usually available through Open-Tran.eu. While no official source of translation memory is available, unofficial translation memories are published in TMX format. It isn't clear which products are involved, and how accurate they would be for languages without official product releases, or during the development phases of upcoming versions of Mozilla projects:

http://www.frenchmozilla.fr/glossaire/TMX/

If you maintain your Mozilla translations in PO format using the Translate Toolkit (see page 11), you can easily reuse your translations in the PO files.

Collection of translation files

If you have some translation files and want to combine them into a single file as a translation memory, you can use some tools from the Translate Toolkit to do that. With pocompendium there are several options for customisation (for how to handle accelerators, for example) and with po2tmx you can create a translation memory file in TMX format used by some translation tools.

For translations in other formats than the popular Gettext PO, there might be converters in the Translate Toolkit (see page 11).

Translation memory and machine translation

Machine translation refers to translation suggestions automatically generated by software. It is often based on some rules of how to translate between languages or a big collection of existing translations. Suggestions from machine translation software could be unnatural, inaccurate and contain grammar errors. It should be said very clearly that **machine translation can't replace human translators**, but are sometimes useful to help translators save time. In reality some machine translation services can provide very useful results in certain cases, although results vary a lot between different languages and topics.

By contrast, suggestions from a translation memory is the result of software suggesting previous work done by human translators. It might be a translation of a different piece of text, and translation software should indicate the difference between your current

original text and the original text of the suggestion. Virtaal highlights these differences to help make a judgement of how relevant a suggestion is and where it should be edited.

Whereas a translation memory will not always have relevant suggestions, machine translation software might always be able to provide some suggestion, even if it is not very good. In some cases it might require less editing to use a suggestion from machine translation software than editing a less relevant suggestion from translation memory. Human judgement is required in both cases to review the usefulness of a suggestion and to correct any possible errors.

Many people are pessimistic about the usefulness of machine translation, and caution in this area is required. It is important for users of machine translation to understand the limitations and to realise that using these suggestions might desensitise them to the proper style required for their language. However, machine translation can be a big help, especially for users who can't type fast.

Reuse in Virtaal

Virtaal can provide suggestions from multiple sources. It can give suggestions from the current file, previously saved translations, and certain network based services. Some of these might be translation memory, and some of these might be machine translation. Make sure that you understand the value and characteristics of each source of suggestions. For more information see page 54 and the Virtaal preferences.

Chapter 7: Technical issues

Translating software is different from translating documents, and this section will discuss several of the most common issues that a localiser should be familiar with. Although some of these issues might seem complicated at first, you can rest assured that you will get used to them quite quickly, and that most translation doesn't involve a lot of such technical issues.

Things to leave untranslated

It is common to find things in translations that should be left untranslated. Easy examples include things like **e-mail addresses** and **website addresses** (URLs) that should be left unchanged. Ensure that everyone on your team knows what e-mail addresses and URLs look like. There are exceptions: example addresses such as http://example.com or user@business.com might simply be intended to show the user what type of input they should provide to the software. If you are sure that these are only examples, it might make them more accessible if they are translated, since a user might realise that it is only an example. In the case where a URL points to a website or documentation that is available in different languages, you might want to change the address to point to the version in your language (if it is available).

We mentioned **brand names** and **product names** on page 26. Ensure that you know what the appropriate handling of these are in your translation.

A slightly more obscure one, is references to **function calls** or other aspects of a network protocol, or programming or query language. Error messages sometimes mention the specific command that caused some error, and these should almost never be translated[4]. For example, spreadsheet software might refer to the COUNT() function – an entry in a spreadsheet cell to count something. There are two clues that this is a function – the fact that it is in capital letters, and the two brackets after the word. Another clue to indicate that something should not be translated, is when some term is enclosed in quotation marks.

4 For some languages spreadsheet functions are translated. You should know if this is the case in your language.

Example

- Incorrect range passed to **COUNT()**.
- Syntax error in **SELECT**.
- Wrong number of arguments to **printf()**.
- Network time-out during **POST**.
- Invalid “**errno**” value.

Of course, there could be other reasons why something is written in capital letters or with quotation marks, but hopefully this will help to identify several cases.

Another thing that might be unfamiliar to many translators, is **command-line options**. When a program is started by typing its name rather than clicking on an icon or menu, it is often possible to specify extra options to influence its behaviour. Although this is hidden from most users, documentation might refer to such options. These options should not be translated, since a translated version won't be recognised by the program. Users therefore should see the untranslated option, even in translated documentation.

Example

See the program version by running with **--version**.

In the case where an option can receive an *argument*, documentation referring to the argument can usually translate the argument, since it is usually just a place-holder for something else.

Example

Original text	Translation
--playlist=**PLAYLIST**	--playlist=**SPEELLYS**
--add-to=**ARCHIVE**	--add-to=**ARGIEF**

There is a convention to write the argument in capital letters, which should help you to recognise it. Having the argument (such as “PLAYLIST”) translated should help the user to realise that the option (such as “--add-to”) should be used unchanged, and that the argument (the translated placeholder) should be changed to the relevant value. When other help text refers to the argument, it is also customary to use it capitalised in a

similar way. If your language doesn't use capitalisation, you might want to put the argument in quotes to emphasise its role as a placeholder.

Escapes

Escaping is a way to represent special text. An escaped sequence consists of an escape marker followed by one or more reserved characters. Translators don't need to work with this frequently, but it is good to be aware of them. The most common is the backslash (\) followed by a single letter. In most cases you will see from the original text when these should be used, and in many cases you can follow the example of the original text closely.

Types of escape characters

These *escaped characters* have a special meaning:

Escape sequence	Effect	Description
\n	newline	A new line in the text (similar to pressing Enter). It might be shown with the ¶ in your translation program.
\t	tab	Add a tab marker at that point in the text. This is sometimes used to align text in columns.
\r	carriage return	This is usually used together with \n in software for Windows.
\\	real backslash	If you need a real backslash then you need to escape the backslash. The computer will only print one backslash.
\"	double quote (")	In some cases a double quote has a special meaning, and should be escaped if you really want a double quote.
\uxxxx	special character	Special characters like © or ™. Programmers might use \u to refer to the special character. Here xxxx is the hexadecimal or decimal value for the specific Unicode character. You probably want to leave it as is.
&apos;	apostrophe (')	In some XML documents, you might need to represent the apostrophe and some other characters with XML entities. For example: link

Example

What you see	What it means	What you do
…server active.\n Do you want…	The newline appears between two sentences. When the program runs the sentences will appear on different lines.	Put your \n escape in the same place between the sentences.
… print jobs or\n other actions…	This is harder to recognise. There will be a line break between “or” and “other”.	Look at the structure of the whole translation. In this case the author is probably using newlines to balance the text (make the lines appear to be equal in length). Balance your translations in a similar way.

You can usually just copy the escape character exactly as you see it. Check that you have the same number of escapes as the original. If your language doesn’t use double quotes or single quotes like English, you can ignore those (\" \' '). In the case of quotation marks, place the quotes around the same concept as in the English.

Don’t unnecessarily change the escape. For example, do not use **/n** instead of **\n** just because you can’t find **** on your keyboard. Be careful for cases like “File \ Open \ New”. This is not escaping. If they are not in the list above, then they are most likely not escape characters. Also be careful to use the same spacing around the escape characters.

Markup

HTML is used to create web pages. You will also see XML (which looks very much like HTML) in GUI translations. They are both examples of markup languages used to specify some information about text. Be aware of which parts of HTML you can safely translate and which you should leave unchanged. HTML is recognised by tags between angle brackets:

```
<tag>
```

A tag can also contain extra information:

```
<img src="picture.jpg" />
```

Here the img (image) tag contains an extra attribute to specify the file name of the picture to show. Furthermore, tags often occur in pairs:

```
<p> </p>
```

Note the difference between the opening and closing tag.

What not to translate

Do not translate the actual tags. HTML consists of tags which indicate the start and end of a section of text. This text could be a heading, a paragraph, a hyperlink or just a piece of text to display in bold:

- `<h1>A heading</h1>`
- `<p>A paragraph</p>`
- `<a href=bob.html>A hyperlink</a>`
- `This is normal and <b>this is bold</b>`

Some markers just beg to be translated, such as these: <title>, , <body>. Do not be tempted – these need to remain in English.

Reordering or changing tags

In some cases, it might be necessary to reorder the tags. If tags surround some text, it means that the translation representing the text inside the tags should probably also be surrounded with the same tags.

Some inline tags can represent something that should be positioned in the translation, such as an image or a footnote. Take care where you put this in the translation to have the same effect as the original text.

Whenever you work with tags, always ensure that opening and closing tags go together in pairs, and that the closing tags always follow the opening tags.

Attributes – which to translate

An attribute is extra information associated with a tag.

Example

```
<body bgcolor=blue>
```

Here “bgcolor” is an attribute and “blue” is its value. Attributes, like tags, are never translated. However some values can be translated. In the example above the value “blue” should not be translated.

There are only a few attributes of which the values can be translated:

Attribute	Meaning
alt	A textual description of a picture, often used in img (image) tags. This is essential for people with disabilities.
title	A text title that pops up when you hover something, exactly like a tooltip.

In some cases you might also need to change the “lang” or “dir” attributes. This is best left to people with good knowledge of HTML.

Translatable text that looks like tags

Some things that look like tags are not really tags and should be translated. For example: <Error>, <File not found>, etc.

If you are experienced with HTML, you will immediately know which are real tags. If you don’t know, use this rule: If its all in lower case or upper case then it is very likely a tag. If it combines case, it is most likely not a tag. If it contains any attribute such as <font color=blue> it is definitely a tag. And lastly if it's the only text in the message like in “<None>” then it is most probably not a tag.

Simple plurals

English adds **s** to the end of most words to form a plural. You will often see that a programmer simply added **(s)** to the word to indicate that it can be both singular and plural. Here is an example in Tsonga, in this case plurals add/change text at the beginning of the word, not the end. This often looks ugly.

```
Show/Hide Axis Description(s)
Kombisa/Fihla (ti)nhlamuselo ya tikhona
```

Furthermore, the grammar in other parts of the sentence might need to agree with either the singular or the plural case, but can’t agree with both.

Here are the options available to you to deal with these type of plurals:

1. If your language does not use plurals, you can just translate it disregarding the embedded plural.
2. Contact the programmer and hear if it can be improved to use proper plurals (see page 43).
3. Use a similar construct if it works in your language.
4. Abandon the singular and use only the plural form. This is what is done in languages where the singular and plural look too different, and the brackets will just make the translation ugly.

Variables

Variables are placeholders for numbers or some text that will be filled into a message when the program is running. For the software to be able to show a message such as "There are 3 files remaining" it has to have a generic message that will also allow it to say "There are 4 files remaining", "There are 5 files remaining", etc. This often looks like this:

```
There are %d files remaining
```

Here we see that "%d" is a variable that serves as a placeholder for the number that will be filled in later. There are many other ways in which the same effect can be achieved. Here are a few examples:

```
• There are {} files remaining
• There are %(number)d files remaining
• There are %1 files remaining
• There are &number; files remaining
• There are $n files remaining
```

These are all equivalent and will appear the same way in the running program: something like "There are 2 files remaining". In these examples, the variable always represents a number written in digits. The number will therefore never appear in words such as "There are two files remaining". Keep the variable exactly unchanged and don't be tempted to translate "number".

Variables can also be placeholders for other things, not just numbers. Here the variable is a placeholder for the name of a program to open a file with:

```
Open with “%s”
```

which will display something like

```
Open with “OpenOffice.org”
```

or

```
Open with “Virtaal”
```

Sometimes programmers might comment on what the variable represents, or you might need to work it out from context. If the variable has a name (like “number” in some of the examples above) you will usually know exactly what will be filled in. It is useful to remember that “%d” will always represent a number written in digits, and that “%s” usually refers to some text string that will be filled in. Messages can contain more than one variable, and could mix number variables and string variables.

Because variables sometimes give special meaning to certain characters, you might need to use an unusual way if you really want this character in your translation. In some of the examples above, the percentage sign (%) is used at the start of a variable. This is part of the variable and disappears when the program is running. If you really want a percentage sign to show, you have to use the special placeholder “%%” which will display as a single percentage sign in the translated program. This is similar to escaping that was discussed on page 37.

Example

```
The download is %d%% complete.
```

This message will display something like “The download is 40% complete” or “The download is 100% complete” with a single percentage sign shown to the user. In your translation, you have to realise that the “%d” is the placeholder for the numeric value (such as 40 or 100 in the examples above), and that the “%%” represents the single percentage sign in the translated software.

If you are unsure how to deal with these cases, don’t worry: they are rare and a programmer for the project should be able help you.

There are a few important things to keep in mind here:

- The variable itself should not be translated or changed in any way, unless you really know what you are doing. Make sure that you know which characters are part of the variable that needs to be kept in your translation.
- It is your responsibility to move the variable in your translation to the correct position in the message to ensure that it will read naturally and correctly in your language. You might also need to adapt other parts of the translation with reference to what will eventually replace the placeholder. Don't follow the word order of the original sentence blindly.
- When there is more than one variable in the string, you might need to reorder them. Variables with names can usually be reordered and positioned freely, whereas symbolic variables (like "%d" or "%s") might need to be handled with more care, or might not allow reordering.
- Since variables might contain things unusual to your language, such as file names or untranslated error messages, consider surrounding a variable with quotation marks if this is a good way to indicate the unusual text in your language.

There are many smaller issues related to variables. Read more in the section on "Advanced handling of variables" on page 48, and also this article:

http://translate.sourceforge.net/wiki/guide/translation/variables

Proper plurals

Programs do have the ability to handle plurals correctly. When the application runs it will determine which plural form to use based on the number that is being displayed. So in English it would display:

- 0 files
- 1 file
- 2 files
- 3 files
- 100 files
- 109 files
- etc.

Here you see that the text is different depending on the number of items. Languages will vary the word form depending on the number in different ways. You need to know exactly how this works in your language for all numbers. In the case of English, the translation is stored as two strings – one for the singular case ("1 file") and one for the plural case ("files" – all numbers except 1).

Example

```
• %d file was added to the archive.
• %d files were added to the archive.
```

Notice how the noun is different, and the grammar adapts to agree with the noun ("was" / "were").

If your language does not have plurals, you will simply translate the two English strings with one string. This is common for some languages in Asia. If your language uses singular and plural nouns like English, you will make a translation of the two strings by also entering two strings – one with the singular and one with the plural. If your language uses more than two noun forms, a separate string for each noun form should be entered to allow a correct message to be displayed to the user depending on the number of things mentioned.

Since an unknown number of things are involved, proper plural messages always make use of variables (see page 41). In the example above, the variable "%d" represents the unknown number. Always remember what the placeholder stands for, and consider reordering things in the sentence to correctly reflect your language.

On rare occasions programmers might ask you for more information about how plural forms work in your language. The following page probably contains what is needed:

http://translate.sourceforge.net/wiki/l10n/pluralforms

Accelerators

An *accelerator key* is a key on your keyboard that you can press to quickly access a menu or function. It is also sometimes called a *hot key*, *access key* or *mnemonic*. If you look at the menu bar on any application you will see that the first letter of each entry is underlined. Notice that File, Edit and View each have the first letter underlined. To

quickly open the File menu press Alt+F. You will notice that most of the menu entries also have accelerator keys, to access Open simply type **o** after typing Alt+F.

An *accelerator marker* is the special character we use to mark accelerator keys when we translate.

Identifying the accelerator

In translations accelerator keys are shown by various marker characters:

Application	Marker	Name	English Text	Displays As	Note
KDE, QT and wxWidgets	&	ampersand	Save **&**As…	Save As…	
GNOME and GTK	_	underscore	Save _As…	Save As…	
LibreOffice and OpenOffice.org	~	tilde	Save ~As…	Save As…	
Mozilla	&	ampersand	Save **&**As…	Save As…	If using moz2po
Windows .rc	&	ampersand	Save **&**As…	Save As…	Also used in Wine

In all of the above examples pressing **A** would take you to the *Save As* dialogue box.

What happens if you want to use the & character without making it an accelerator? In this case the accelerator is usually escaped by using && (for KDE) or & (for Mozilla): “Mail && News” or “Mail & News”. As a translator you would be free to drop the & and use the equivalent of the word “and” in your language.

Selecting

How do you select an accelerator key for your languages? Keep these principles in mind:

1. Try to keep it similar to the English. This makes it easier for users to switch to localised interfaces.
2. Accelerators should not degrade readability too much.
3. Avoid duplication. Ideally, in any menu, window or dialogue box, a letter should only be used as an accelerator for one string. If the letter 'T' is used for two menu entries, it is usually not a big problem, but the software isn’t as usable as it could be.

We usually follow these simple rules.

1. Try to keep the same letter as the original. “&File” → “I&fayile”
2. Try to keep the same position as the original. “&File” → ”&Ifayile”
3. Select an uncommon letter. This might avoid clashing accelerators.
4. Try to use a letter that is close on the keyboard to the original letter, therefore being easier for the user to adapt to.
5. Avoid narrow letters like **i** and **l**. It might be too hard to see the underlining under them.
6. Avoid adding accelerators under characters with descenders such as lower case **g**, **j**, **p**, **q** or **y**. The underlining might strike through the character and cause the character to look odd and not be immediately recognisable. For example in KDE, &g would appear as g which looks messy. It is best to confirm with an actual test, since some rendering systems actually support underlining clearly when the letter contains a descender.
7. Try to avoid the letter next to a descender for similar reasons as above.
8. Make sure the character is actually available on the user's keyboard. Accented characters such as **ä** might or might not be present on a keyboard in the language. It might even be difficult for some users to type unaccented characters like **ɔ** and **ɛ**.
9. If your language uses diacritics under certain letters such as **ṭ**, **ḍ̲** or **ọ**, avoid these characters as well, as the underline might run through the diacritic. Avoid letters with diacritics entirely unless you are very sure that they will be usable by the end-users.
10. If your language uses advanced input methods to type (like some Asian languages), you can’t use any of the characters in your translation. Use the original English accelerator: “&File” → “XXXX(&F)”. In some software you can ignore defining the accelerator in which case it will default to the English version.
11. Non-letters like numbers and punctuation marks can be used if they are present on a standard keyboard as is common amongst the speakers of your language. This will usually only be a relevant choice if it was chosen for the original text as well.

Accelerator Clashes

If we have two accelerator keys using the same letter, we say that we have a clash. Assume that the following list are the top three entries in a menu:

- Author
- Address
- Available Actions…

You will notice that all the accelerators use the **A** key – this is a clash. Fortunately most applications will cycle through the options as you press A repeatedly.

Here is a better choice of accelerators for the same menu:

- Author
- Address
- Available Actions…

We now use **A**, **d** and **v** – there are no conflicts.

Examples

English text	Good translation	Bad translation	Appearance in original	Appearance in translation	Notes
&File	&Faele	F&aele	File	Faele	Rule 1, 2
&View	Le&belela	&Lebelela	View	Lebelela	Rule 3, 4
~Tools	Di~tlabela	~Ditlabela	Tools	Ditlabela	Rule 1, 3
&Help	T&hušo	&Thušo	Help	Thušo	Rule 1
_Move	Š_uthiša	_Šuthiša	Move	Šuthiša	Rule 8
Zoom &100%	Godiša &100%	&Godiša 100%	Zoom 100%	Godiša 100%	Rule 11
~Font Size	Bogola bja ~fonte	Bogola bja~fonte	Font Size	Bogola bja fonte	Spaces

Review

Review your choices in the running application. Clashes are not a big a problem, and can sort themselves out over time. Refer to the section on “In-product review” on page 71.

Advanced handling of variables

A slightly more advanced use of variables occurs in some cases to allow the translator to customise the way that the variables are displayed. The examples and techniques in this section are a little bit more advanced. It isn't important to understand all of these immediately, but understanding these issues might help you to solve some problems in your translation and improve the quality of your localisation work.

Reordering

Your translation might just work best with the variables in a different order from the original text. Doing this is easy if variables have names, but it is common for programmers to use unnamed or symbolic variables such as "%d" and "%s".

```
There are %d photos on the device %s
```

This might display as "There are 55 photos on the device Nikon XYZ". In your translation, you might prefer to reorder the variables, to say the equivalent of "The device Nikon XYZ has 55 photos". You might be able to do it this way:

```
The device %2$s has %1$d photos
```

The variables are now changed to reflect their order in the original English string. The variable "%d" was initially first, so now it is represented by "%1$d" now that it isn't used first any more. In lots of software (especially in C like languages using the printf() function) you are able to do this. The programmers would be able to confirm.

Omitting

In some cases it might be possible to entirely leave out a variable in the translation. This can be useful for the correct translation of certain plural forms, or if an entirely different wording will be used in the translation. This won't be possible in all situations, so check with the programmers if it is safe to do.

In some cases where it isn't possible to entirely leave out a variable, it can be added in a way that will prohibit it from displaying. A "%s" variable can sometimes be represented with "%.0s" which would stop it from displaying at all.

Times and dates

Variables are sometimes used to customise the display format for times and dates. Check the section on "Number and date formats" on page 76 for more information. In this case, you might need to use different variables from what was used in the original text.

Example

Here is a string used to format time to look something like "8:15 pm":

```
%I:%M %p
```

This is the usual American 12 hour format with the extra "pm" or "am" indicating if it is in the morning or not. In a language where a 24 hour format is more appropriate, the translation should be:

```
%H:%M
```

This will display the same time as "20:15" with no "am" or "pm" at the end. Different variables are used, and there are even a different number of variables. In the case of such time and date formatting strings, this is correct.

The exact details of how these variables work might depend on the programming language used by the project, although they are often the same. Here are a few pages for popular programming languages in FOSS projects:

- **C / C++**
  http://opengroup.org/onlinepubs/007908799/xsh/strftime.html
- **PHP**
  http://php.net/manual/en/function.strftime.php or http://www.php.net/date
- **Python**
  http://docs.python.org/library/datetime.html#strftime-strptime-behavior
- **.NET**

http://msdn.microsoft.com/en-us/library/system.globalization.datetimeformatinfo.aspx
On some systems you might also be able to consult the manual for strftime, which should correspond to the first link above.

Formatting

In some cases you can specify the exact format in which variables should display. This might be useful for the correct formatting of numbers in accounting software, for

example. You can control if the sign (+/-) should show, if some zeros should be added to the front, etc.

This is not frequently required, and you might never need to bother with such formatting. For more information, see the possible choices for formatting variables:

http://opengroup.org/onlinepubs/007908799/xsh/fprintf.html

Chapter 8: Localisation tools

We introduced the two main localisation tools discussed in this book earlier on page 11. Virtaal and Pootle complement each other and together fulfil most of the functionality a translation team should need. Pootle allows translation online and offline with a focus on volunteer contribution. Virtaal is a powerful translation tool focused on simplicity and productivity, and works well with or without an internet connection. You can always download files from Pootle for translating offline in Virtaal if you prefer.

Virtaal

Virtaal is a powerful translation tool, yet simple to use. You can increase your productivity by ensuring you know all the shortcuts and tricks and without being distracted by a cluttered interface.

Although most features are available using the mouse, Virtaal is designed to encourage you to work as much as possible with your keyboard to increase your speed and keep the translation fun.

Opening a file

Mostly you should be able to simply open a translation file by clicking on the file in your file manager. The file might be associated with another program in which case you can look for Virtaal in the context menu by right-clicking on the file.

You can also run Virtaal and open a file with File→Open or Ctrl+O.

Virtaal supports many popular file formats in the FOSS world.

Normal translation

After opening a file, the first translation will be shown with your cursor in the field below the original text. You can simply type your translation and press <Enter> when finished - just like in your word processor. Note that <Enter> moves you to the next position where

you want to type. In the case of a translation with plurals, enter will take you to the next line in the same translation.

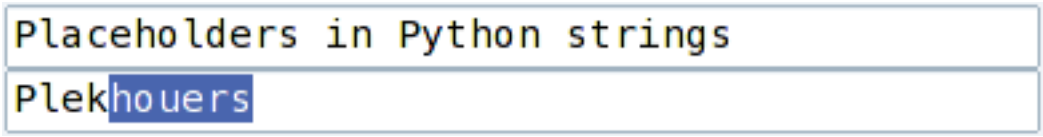
If you have the correct spell checkers installed, spell checking should be active for both the original and the translation.

You can undo normally using Ctrl+Z.

Time savers

Auto-completion

Virtaal will save you some time by trying to complete some long words for you. You will see the auto-completion suggesting a possible word, and the suggestion can be accepted by pressing <Tab>. If the suggestion is not what you want, you can simply continue typing the word you had in mind. If you accepted a suggestion that you don't want, you can simply undo normally with Ctrl+Z.



Auto-correction

Virtaal will save you some time by fixing certain common typing mistakes or spelling errors. How mistakes are corrected depends on your language, and there might possibly not be information for your language yet. Feel free to get involved in the project to improve this feature for your language.

If Virtaal automatically corrected something which you didn't want, you can simply undo the step with Ctrl+Z.

Copy original to translation

Sometimes it is easier to have the original string as a start to only replace a few translatable elements. Translations containing XML markup or many variables might be more work to type again than to just start with the original text. You can easily copy the original text into your translation area by pressing <Alt+Down>.

For some languages, you will see how Virtaal automatically changes the punctuation marks to fit the conventions of your language. This could involve quotation or other punctuation marks, or the spacing between certain elements. For example, a "quotation"

automatically becomes a « quotation » in French, without the translator having to change the quote characters or the spacing.

If you don't want the changes to the original text that Virtaal automatically did, you can simply undo the step with Ctrl+Z.

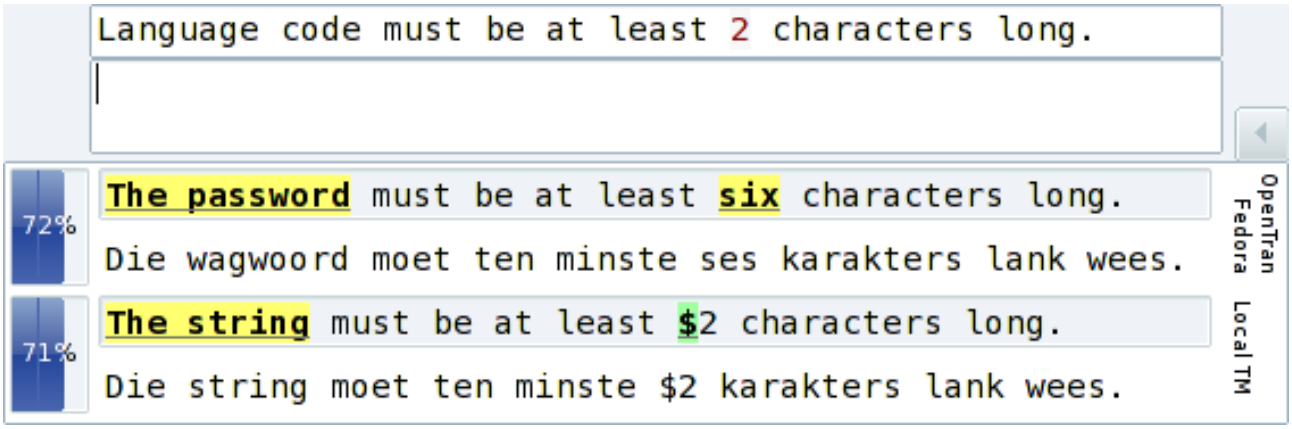
Copy a placeable to the translation

Placeables are special parts of the text that can be automatically highlighted and easily inserted into the translation. You will see that certain parts of the original text will be highlighted. To select which placeable to insert, press <Alt+Right> to move the highlighting to the correct placeable. You can insert the currently highlighted placeable by pressing <Alt+Down>. After you have inserted a placeable, the next placeable will be highlighted.

Copy a term to the translation

Highlighted text will show which terms Virtaal recognised, and allow you to handle them as placeables. You can use <Alt+Right> and <Alt+Down> the same way as with other placeables. If there is more than one suggestion for a term, Virtaal will display the choices in a menu. Select the translation you want, or press <Escape> to continue typing.

Use a suggestion from TM or MT



If Virtaal has a suggestion obtained from translation memory or machine translation, it is displayed underneath the editing area. You can put the first suggestion into the translation with Ctrl+1, or use Ctrl+2, etc. to select the others. You can also double click the suggestion to obtain the same effect.

Navigation

Above we saw how we can easily advance to the next point of translation by pressing <Enter>. You can also move around easily between rows with <Ctrl+Down> and <Ctrl+Up>. To move in large steps, use <Ctrl+PgDown> and <Ctrl+PgUp>.

Virtaal will move you between certain rows. Normally it will move between all rows, but other ways of navigation can be selected at the top of the Virtaal window.

Incomplete mode

If you activate the "Incomplete" mode, Virtaal will move between untranslated and unreviewed translations. This allows you to quickly find the places where you need to work. Translations will still appear between the same rows in the file so that you can see the context that you are translating in.

Searching mode

If you activate "Search" mode or simply press <F3>, Virtaal will move between all the rows that correspond to your search query. Translations will still appear between the same rows in the file so that you can see the context that you are translating in.

To move back from the search box to your translation, simply press <Enter>, or go back to another mode.

Quality checks

If you activate “Quality Checks” mode, Virtaal will move between the rows where selected quality checks indicate a possible problem. This is a powerful way of reviewing translations for quality by testing for several issues that can affect the quality of your translations. Refer to the section about “Automated review” on page 69.

If Virtaal indicates a possible problem with a translation, it doesn't mean that the translation is necessarily wrong, just that you might want to review it. You should also select the correct project type (GNOME, KDE, Mozilla, etc.) in the project type selection. This will improve the accuracy of the quality checks.

http://translate.sourceforge.net/wiki/virtaal/checks

Pootle

Pootle is a popular web-based system for translation and translation management. It is used by several FOSS projects to manage their localisation work. Pootle allows online translation, and it has several features to help with review and team organisation. It is ideal for team work, casual contributions, and translation sprints (Translate@thons).

A Pootle server can host one or several projects for any number of languages. Some servers are used to translate the software of one project into many languages, whereas others are used for coordinating the team work for a single language team on several localisation projects. Pootle allows team work with different roles for different team members. Pootle users can also make use of several features to help them improve their quality, such as terminology assistance, and quality checks.

Since Pootle is a web-based system, it is installed by server administrators, and the activity allowed on a Pootle server, is determined by them. To translate anything not yet on the server, contact the administrators. Realise that each Pootle server might have its own purpose and way of assigning permissions to volunteers.

Most Pootle functionality requires that you log in with an account.

Finding your work

From the front page of a Pootle server, you can discover the available projects and languages on the server. A summary of the progress is shown to get an impression of the completeness and activity for a specific project or language. Once you have selected the

language and project, you will see more detailed statistics about every file in the project, and have access to more functions for translation and review.

If you want to translate, the “Translate” tab provides access to the features for both online and offline translation. To quickly go to the incomplete parts of the project, choose “Quick Translate” for the file or directory. To translate offline, download the appropriate file or files. After completing the work offline in Virtaal, use the upload form at the bottom of the page to submit the work back to the Pootle server.

To review work (even your own work), visit the “Review” tab. It provides access to suggestions left by team members as well as quality checks that might indicate possible places of improvement in the translation.

If you are the administrator for this project or language, you might also be able to configure permissions and perform certain file management work. Refer to the Pootle documentation for more information on these features.

Server administrators can add languages, projects, and also provide translation templates (refer to the section on “A simple localisation process” on page 59 to understand the role of templates). There should be contact details to allow Pootle users to contact the server administrators.

Terminology

Pootle can help translators with terminology. Terminology can be specified to be global per language, and can be overridden per project for each language. A project called “terminology” can contain any files that will be used for terminology matching.

Project administrators can generate a list of frequently occurring terms from the “Translate” tab in the Pootle interface, which can help to quickly standardise some frequently occurring terms. It is also possible to add extra entries.

http://translate.sourceforge.net/wiki/pootle/terminology_matching

Suggestions

Pootle has the ability to optionally allow users to provide suggestions that need to be reviewed before they are accepted into the real translation files. Who is allowed to do what is determined by the configuration of permissions for the project or the server.

This allows for a team to form with different roles for different team members, and makes it possible to have a more explicit review step that requires suggestions to be checked before they become the real translations. This also allows the collection of alternative translations.

http://translate.sourceforge.net/wiki/pootle/suggestions

Quality checks

Pootle provides a powerful way of reviewing translations for quality by testing for several issues that can affect the quality of your translations. Refer to the section about “Automated review” on page 69.

If Pootle indicates a possible problem with a translation, it doesn’t mean that the translation is necessarily wrong, just that you might want to review it. Pootle administrators should indicate the correct project type (GNOME, KDE, Mozilla, etc.) in the administration pages. This will improve the accuracy of the quality checks.

http://translate.sourceforge.net/wiki/pootle/checks

Searching

Pootle provides a searching functionality that allows translators and reviewers to search through translations for some text. The search box is shown close to the top of the page. Searching can be used to find specific things you want to work on, see how issues were solved before, or to verify consistency in your translations.

Search results are up to date, and will reflect the current translations in Pootle.

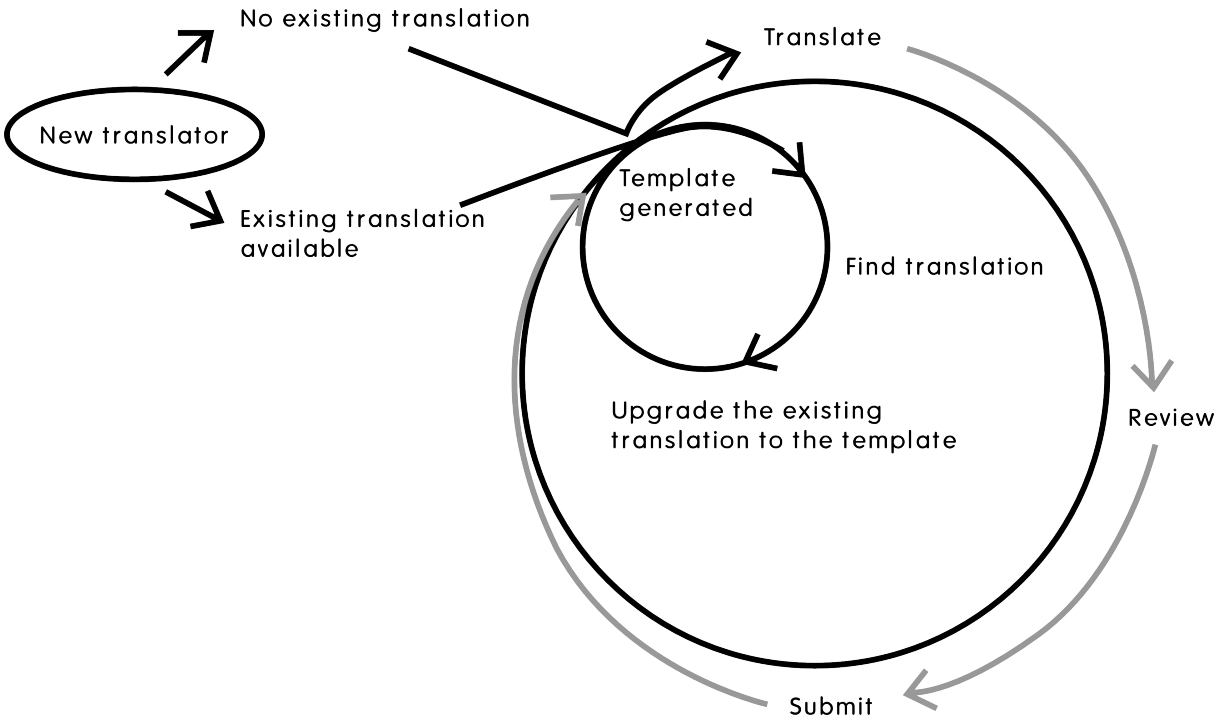
Chapter 9: Localisation projects

A simple localisation process

In the simplest case, you might be offered a single file to translate, open it in your translation program such as Virtaal, and send it back to the project when you finished the translation and review. Although things are usually more complex than that, this is realistic for some projects, especially when the program is translated into your language for the first time. You can also keep things simple initially by working on projects with a simpler process. In our case studies, you will find a description of the project Tux Paint, which is very simple and an ideal starting point (see page 79).

In the general case, however, you might need to perform extra steps, or you might need to create your own files to translate following some prescribed steps. In the case where there is an existing translation, you will need to update the translation file to contain all the newest strings to translate, so that it corresponds to the program version you are translating.

A lot of projects follow a process similar to this:



- A translation template is generated from the source code (programming files) containing all the translatable text.
- New translations simply start from this template. Existing translations are upgraded to conform to the new template.
- Translators translate the file, often also performing some review stages.
- The finalised version is submitted to the project, and will form the start of any future work in this language.

Note that you should never need to start translating from scratch if some previous translation is available.

Finding the relevant files

To translate an application, you will need to find the files to work on. Hopefully the project has documentation specifically for translators explaining exactly what to do, including how to submit your work to the project. For a project that use a Pootle server, this might be absolutely straight forward. However, even if they don't, things might be quite predictable.

Many projects might simply make a single file per language available for download. Maybe all you need to do is to download the file, translate and test, and e-mail it to a developer. There might be more specific instructions about submitting, such as creating a bug report and attaching your translation.

Some projects might expect that you interact with their *version control system*. A version control system is a system used to keep all files for the projects, and therefore usually also the translation files. Most software is developed with such a system in use, and in the case of FOSS, it will usually be publicly accessible. This might be the preferred way to obtain files, and to submit them back to the project. In order to submit a file to the version control system, you will need to be authorised to do so, and each project might have its own way of doing this. In most projects it isn't compulsory to understand the version control system in order to localise the application, but in some cases it might simply be best for you to become familiar with some of the basics of how it works. The programmers should be able to help you get started and recommend any required

documentation. Some basics are explained on page 64 in the section on “Version control systems”.

In whatever way you obtain your files, it is important to know whether the files are already up to date, or whether you are required to “upgrade” them to contain the newest work. This is one of the reasons why you should usually **not** start by translating from a released version of the software, since that won't give you up to date files.

For most FOSS projects you will be required to translate a single file. The most common format is .po, although several others exist, such as .ts, .properties and .php. The files are likely found in a directory called “po”, “locales”, “lang”, or something similar.

Many projects are dependent on translations done outside of the main project, such as those done by software libraries. For a fully translated application, it might be necessary to complete these translations as well. Ask the programmers if you are in doubt.

In some cases there might be a special procedure for adding a new translation. For example, it is common to put a line with your language code in a file called LINGUAS. There might also be a convention to add an entry to the ChangeLog for each change to the software. This differs from project to project, so check the project documentation.

In the case where the project provides files that can't be opened directly in translation tools, you might need to look into converting these files into a translation format. In many cases, this is important, and have many advantages. Read more here:

http://translate.sourceforge.net/wiki/guide/monolingual

The Translate Toolkit has several converters used by many FOSS projects to translate their files (see page 11). Several other converters also exist, such as po4a and xml2po.

http://translate.sourceforge.net/wiki/toolkit/index

http://po4a.alioth.debian.org/

Project communication

You will be required to interact with people while you are involved in a project. This section describes some of the ways that FOSS projects commonly communicate and transfer information between people.

E-mail

A *mailing list* is a system that distributes e-mail to subscribers to facilitate group discussion by e-mail. Any e-mail sent to the mailing list address is automatically distributed to all subscribers. A lot of projects use mailing lists for communication between project contributors. There might be a single list for programmers, users and translators, or several mailing lists each with its own purpose. You will often need to join a mailing list to meaningfully take part in a project. It might be the best way to be notified of upcoming releases, have your questions answered, and get the know more about the project.

Note that a lot of FOSS projects prefer such mailing lists to keep to the intended topic. Any off-topic discussion, advertisements or purely social use of a mailing list will be unacceptable in many projects. There might also be conventions about whether attachments are allowed in messages to the list, etc. As you receive some messages from the list, you might notice some of the conventions. You should never add the mailing list address to other services like social networks.

Remember that most of these mailing lists are usually archived and publicly available. Any e-mail you send will show your e-mail address, and the content will be available to anybody with web access. Always stay polite and refrain from arguments. Keep in mind that the members of these lists are usually from different backgrounds, and people will have different cultural values and abilities in English. Do your best to be very clear in what you say or ask.

It is usually not considered a good idea to contact any specific project member directly by e-mail, unless this is an indicated way to do something (like submitting your translation) or if you get to know the person involved well. Therefore you should always use the mailing list address for answering, and thereby ensure that relevant discussions stay on the mailing list for everyone's benefit.

Chat channels

Several FOSS projects use chat services such as IRC as an additional communication medium. It is sometimes used for user support, or for more interactive discussions. It might the best way to get a quick answer to a question, or to learn some procedure that might involve a few questions and answers. Although chat programs are often the recommended way to join these channels, you might also be able to use your web browser.

Although several people might seem to be in a channel, it might be that they are not actually at their computer and reading your message. Always be patient for an answer, and stay in the channel while waiting for an answer. It is not uncommon for people to get back to their computer after lunch or a meeting and answer you after an hour of waiting. People might even be in another time zone than you are, and might only see your question after waking up.

Just as with mailing lists, you should be polite, and it is possible that communication in the channel is logged and publicly available. When you have a question to ask, simply ask it. It isn't necessary to ask if you can ask a question. In most channels the participants would prefer that you state your question up front with all relevant information.

Never paste large amounts of text in a channel. Some servers might cut your connection, and it is not a pleasant way to work with many lines of text. Better ways might be e-mail, or a *pastebin* – a web site where you can paste text and then only use the link in the chat channel.

Bug trackers

Many projects make use of a *bug tracker* such as Bugzilla or Trac – a web based system where problems, requests, and improvements to the software are planned and discussed. A link is usually provided on the project's web site.

You might be expected to submit your translations in such a system. Additionally, this might be the best place to report any errors you find in the original text that you translate. Although there are often many fields, don't worry about them. You can usually just leave any difficult ones at their default values. In time you might learn each project's preferred way of filling in these forms.

Wikis and forums

Wikis are sometimes used to build a project's website, project documentation, or even instructions for participants. The idea of a wiki has become well known due to Wikipedia. You might want to get involved in translating the project documentation or website, and some of this might happen in wikis. Spend a little bit of time to ensure that you are familiar with the wiki syntax used for the project wiki (how to format the text).

Forums are not very common in FOSS projects, but are sometimes used for user support or discussions among participants. You might want to help out with user support in your language when you are familiar with the software.

If we believe that software localisation is important, obviously documentation and support in the user's language is also important. For small languages there might never be enough people to help with all these things, so remember to prioritise your work and keep your goals in mind.

Version control systems

We briefly mentioned version control systems in an earlier section. Although there are many things to learn about version control systems, you will probably only need to know a few basic operations to contribute translations. Find out which version control system is used in the project, and find out how to do the following:

- Get the complete project with all files from version control
- How to update the files at a later stage to the newest changes made by others
- How to submit a file

If you can do these things, you will probably be able to do most of what is expected from you. Although all changes can be undone in a version control system, you shouldn't be careless or make any unnecessary changes. Ensure that you know the following:

- When you are allowed to submit a file
- If any specific commit message is expected
- Whether there are any other changes you might need to make (such as a LINGUAS file)
- If you need to do any specific preparation before you commit, such as switching to a different branch

Branches are usually a way of doing work on two different versions of a product at the same time. You might need to only submit your translation to one version of the product, and therefore only to one branch in the version control system. Ask the programmers if you are unsure.

Some of the principles of version control systems are explained here:

http://betterexplained.com/articles/a-visual-guide-to-version-control/

Project schedule

Each software project has its own schedule and pace of development. If a project releases new versions very often and accepts translations at any time, you can probably contribute your translation whenever it suits you. For projects that only release on certain occasions, you want to ensure that your work will be useful to as many people as possible, so you should try to think strategically about your participation in the project.

It is common for projects to announce their release plans in advance and ask translators to update their translations. This is an excellent time to contribute your translation, as it is probably an indication that the developer is actually thinking about translators, and is trying to create good circumstances for them to do their work in. If you are starting with a new translation of a project, you should take extra care to ensure you have enough time to complete the whole task and do the testing. Read more in the section on “Counting and estimating” on page 73 and about “Testing and review” on page 69.

If you are translating a really large project for the first time, you probably need to start long before such an announcement anyway, which means that you are somewhat less affected by it for the biggest part of your work.

Such an announcement from a developer coincides with a phase called the *string freeze*. This means that the text that needs to be translated is finalised for the release, and no further changes are planned for the text. This means that you can do your translation work knowing that you are translating the final text and probably won’t need to adapt your translations to changes in the original text until the release happens. Such an announcement might also specify which software version will be released, and maybe which branch in the version control system should be used. If you are unsure, ask the relevant developers – you don’t want to spend your time translating the wrong version!

If you start too late, you might not complete the work, or you might be too late for your contribution to be accepted into the release. Starting earlier is usually safer, but not necessarily better. If you start before the string freeze, you might need to make extra changes when the final string freeze is announced. If you stay up to date with the news in the project, you can avoid disappointment.

FOSS issues

Motivations

As mentioned earlier, people will have different motivations to contribute to FOSS projects. Some developers do it as a hobby, or to solve some problem that they wanted to solve. Some contributors work for companies who contribute to FOSS projects for several reasons. The important thing to realise is that many of the people you encounter in a FOSS project might be involved there for different reasons than you are. If you want to get help from someone, it might demand some of their time, and they might not be able to help you right away because they are at work. Someone might not be interested to help you because they are not interested in the specific issue you need help with. Usually people are quite helpful, but it is useful to keep in mind that very few people have an obligation to help you. Therefore always be polite, especially when asking for help. Also try to help other people in the project when you can. There is a big culture of doing favours in FOSS projects, so remember that they are favours, and try to do the same to others.

Licences

Software is distributed under certain conditions which determines what you are allowed to do with the software. These conditions form the licence which is often shown when installing software, or may be in a file in the source code of the software. Of course Free and Open Source Software is distributed under very attractive licences allowing us to do many things with it, including localisation. There are a few issues that deal with licencing and copyright that might be relevant to localisers.

Firstly, you have to understand that your contributions to the project will be distributed under the terms of the licence used for the rest of the software. You might want to familiarise yourself with these terms to ensure that you know what other people are allowed to do with your translation. For example, most FOSS licences allow people to sell copies of the software, and it is best if you are not surprised when you see this for the first time!

To contribute to some projects, they might require you to sign a *copyright assignment form*. The project might do this for different reasons such as wanting to know your real identity, enabling them to relicense the software at another stage, or to be able to act on

your behalf in legal disputes. While this is not very common, it can be encountered in some projects.

In some cases, you might not have the rights to distribute your own translated version while maintaining the original branding (like the name and logos) of the software. Enquire if the project has any requirements with regards to branding.

Keep in mind that projects are free to reuse your translation according to the terms of the licence, and that you can't retract the rights you gave to the project when you submitted the translation. It is even possible that the project could appoint someone else for translation in your language who will change some of your work without your permission. Although very rare, this might be required to keep the project alive. This should almost never impact your work.

For more reading, see Free/Open Source Software Licensing by Shun-ling Chen.

http://www.iosn.net/licensing/foss-licensing-primer/foss-licensing-final.pdf
http://en.wikibooks.org/wiki/FOSS_Licensing

Wider involvement

An interesting aspect of FOSS projects is that contributions are usually welcome in all activities of the project. Translation, documentation, testing, marketing, programming, design – these are all useful ways of contributing to a project. You might only be interested in translation, but you might find that your involvement in many other activities can be very useful; not only for the project, but also for your localisation work.

Reviewing English text is very important for English users, but also to ensure that translators are working from clear original text. If your English is not very good, you might still be able to point out which things are hard or impossible to translate, thereby helping to improve the original text for your own translation work. As a translator you are probably one of the people who read the English text with the most attention, thereby making yourself a good proof reader of the English text. You can usually report errors with the original text in the bug tracker for the project.

Testing the software with the translated interface is an important part of your localisation work. Testing is discussed later on page 69. A valuable side-effect of this is that you can make a useful contribution to testing the software itself in preparation for a release. You are also the best person to test compatibility with anything related to your language or country, such as handling of text in your language, compatibility with local websites, font support, etc.

Designing the software for a diverse user group all over the world isn't always easy, and this is another way where you can help the project to ensure that the designs take your language and culture into account. It could be as simple as asking for more space for a string with a very long translation in your language. It could also include more far reaching design decisions such as features that are required for your language that the developers might not have been aware of.

Marketing the software all over the world sounds like a lot of work for a small group of volunteers, except if a project has contributors in many countries. You can become an ambassador for the project in your country or geographical area, and even in this case, it could often involve translation. Marketing material, websites, brochures, etc. might be available for translation, and the project might depend on you to ensure their success in the place where you live.

Involvement in events is not a frequent activity for many contributors to FOSS projects, but you might get an opportunity to represent a project you are very familiar with at a local event. This is a great way to promote the software, and maybe to gain more contributors for your localisation work.

Chapter 10: Beyond mere translation

In earlier sections several important issues in FOSS localisation are discussed. With some time and practice, these should all become easy and natural for you to do. You can improve your localisation work even more with some of the topics discussed in this section. This should help you to spend your time more effectively, and work at higher quality.

Testing and review

Testing your localisation is very important. It is natural to make mistakes, but too many mistakes in your localisation will leave a bad impression with the users, and put them at a disadvantage for using the localised version of the software. This section explains some different ways of testing.

Automated review

For the purposes of this book, automated testing refers to using software to automatically review our translations. Since software is really good at simple repetitive tasks, it is ideal to help us discover things like spelling mistakes, errors in capitalisation, unwanted spaces, as well as mistakes with variables and markup. Several of the issues we discussed in “Technical issues” on page 35 might be detected by automated review tools.

Pootle and Virtaal incorporates several quality checks that will help you to find certain types of errors. Some of these are only of a cosmetic nature (such as spaces and punctuation). Others are a bit more serious (such as a missing web address or suspiciously short translation). Some tests might indicate serious issues that could crash the software or prevent some functionality from working (such as errors with variables).

Read more about all these quality checks here:

http://translate.sourceforge.net/wiki/toolkit/pofilter_tests

It is important to realise that these tests could suggest that some translations contain mistakes when they are indeed valid. Don’t follow the advice of these tests blindly, and

don't let their results encourage you to do something incorrect or unusual for your language. The tests are just tools intended to help you. We have found them of crucial importance in our work, especially with inexperienced translators, although even the most experienced translators sometimes make mistakes.

Automated testing is cheap in the sense that it doesn't require much of your time. It quickly shows you where to spend your time to fix a possible problem. Always review the quality checks – it is an easy way to avoid certain errors.

Software written using the Gettext translation system usually use a program called **msgfmt** to convert the translation file into the file needed by the software. It also does some basic checks, such as checking the file format, invalid use of variables, etc. Any serious error reported by this tool **must** be fixed. Usually the translation will not be used if there are any errors reported by msgfmt. Software written using the Qt translation system, has a similar program called **lrelease**.

Manual review

Although automated review tools can help us a lot, there will always be room for review by another human translator. If you join an existing team, the existing team coordinator might review all submissions from other team members. You might be able to ask someone to read over your translation file and check for possible areas of improvement. Another pair of eyes can often help to improve the quality. A translation program with spell checker and quality checks is probably the best software to use for this task.

Mature translation teams usually have an arrangement of how work will flow between translators and reviewers, and might even have rules that all files have to be checked by a reviewer. In some projects there are tools available to help you implement such workflows, and to help in communication in the team. The localisation website of the GNOME project is a great example in this regard.

When you are working with PO files, it might be easy enough to simply mark translations as needing work (fuzzy) when you still want someone to review them. The reviewer can then approve the translation by removing the fuzzy marker. Although this might be a bit limiting in some cases, it is a simple way to incorporate manual review into your localisation process.

During review, reviewers should pay attention to spelling, grammar, terminology, consistency, and all the other things that might be important. When you review someone else's work, always try to provide polite feedback to serve as encouragement.

In-product review

A very important step in your localisation process is to review the translations in the running application. This will allow you to see the result of your translation work, and is always exciting to see for the first time. The basic principles of testing is the same, regardless of how the translated application is executed or built. The ways of how to do this can be very different for different programs, therefore we'll discuss only the most common case in the next section.

The purpose of in-product review is to review the translation in the way that the user will experience it. Now it is not just about the language issues, but about the whole experience of the user. To ensure that you test the translation well, you have to try to find every part of the program. Go systematically through every menu, look at every option in every dialogue box. Look for tooltips when you keep the mouse over some part of the program. You can even try to do unusual things, like opening unsupported files or entering invalid values in the configuration to try to see error messages. Even the error messages should be translated and reviewed!

Important issues to check for include things like consistent style and terminology, whether references to buttons use the same text as the buttons themselves, and if translations make sense in the context where they are used on the screen. Remember that programs might sometimes combine translations from several files, so consistency isn't just about the translation quality in a single file.

Another important thing to check for is any strange layout caused by very long or very short translations. It could be that the programmer didn't anticipate a translation as long or as short as yours, and the user interface might not appear neatly any more. Most software in the FOSS world will try to adapt automatically to changes in the translation by resizing things as necessary. In a few remaining projects it is still necessary to resize these things yourself. This is a time consuming task that requires you to spend much more time on testing, and might also require that you test the software on multiple platforms.

Another important thing to test in the running application is the use of accelerators. We discussed accelerators earlier on page 44. Now you need to check that the accelerators were chosen well, and make any necessary corrections. In each menu, and each dialogue box, it is important to test every accelerator. It is usually activated by pressing Alt and the underlined letter. It might be necessary to press Alt to see the characters underlined. Does the relevant letter activate the control correctly? What happens when you press the

letter again? If two controls on the screen use the same letter as accelerator, it often happens that they are alternately activated when you press that key. This is not a big problem, but should be corrected in your translation if possible. In big menus and dialogue boxes, you might need to plan carefully to select unique accelerators for each control. However, remember that the most important functions, especially in the main application window is probably the most important for you to review, and to ensure that they work well. Also try to keep the accelerators consistent for common things (like “File” and “Help”) between applications.

If you translated documentation, you should review it in the help viewer. Check that page titles and links between pages are correct, and that any search features show the correctly translated entries. Ensure that the help text uses the same terminology as the application. Reviewing the translated content itself is similar to reviewing translated documents.

Testing a program using Gettext

Many FOSS projects make use of software called Gettext to support many languages in their project. In such projects you would have translated a PO file, and need to generate an MO file for testing. Some translation programs might be able to do this easily. Otherwise you can use the program **msgfmt** from the command-line like this:

```
msgfmt  -cv current_translation.po  -o program_name.mo
```

This will perform some basic tests on your translation file (refer to page 70) and mention any serious issues that still need to be fixed. If nothing serious was detected, it will generate the file “program_name.mo”. The correct name will depend on the application (look for the current .mo file in your or other languages), but is usually just the name of the application in small letters, followed by “.mo”.

Now copy the new .mo file you created over the existing file (after making a backup), or copy it in the correct location. Here are the most common directories that are used (replace 'xx' with your language code):

- `/usr/share/locale/xx/LC_MESSAGES/` **(most Linux-based systems)**
- `C:\Program Files\Program Name\share\locale\xx\LC_MESSAGES\` **(Windows)**

You might need to create the directories if they are missing (especially on Windows). Remember that your system language must be set correctly.

Prioritisation (when you can't do everything)

As much as we might want to have all software in our own language, we will always have too little time, too few people to help us, and the amount of software will just get more and more. Does that mean we should give up? Of course not. It has been shown many times in the FOSS world that a small team or even an individual person can do a lot of good localisation work and have a lot of translated software available. This section explores some ideas of how to maximise your impact, and tries to provide an idea of what to do if you can't do everything. Remember to keep your goals in mind!

Counting and estimating

If there is too much work to do, or you think there is too much work to do, it is very useful to know exactly how much work is involved in translating a certain file or piece of software. It is common in many FOSS projects to count the number of strings (messages), but in reality this doesn't give a very complete view of how much work is required to finish a certain task. Counting the number of words is a very good indication of the amount of work required to complete a translation.

Some projects might give a summary of the progress in each language. Always check if the numbers they mention are messages or words. Most translation tools can also give such a summary. Pootle and Virtaal prefer to mention word counts rather than message counts. You can also count many files easily on the command line with **pocount** from the Translate Toolkit (see page 11).

Of course this doesn't say anything about the difficulty, the amount of reuse you might get from a translation memory, or the quality of the English. These and other issues will

still influence how long a translation takes, but the information will help you to make better decisions.

Choosing the product or project

Depending on your goals, you might already know what software you want to translate. If you are still deciding what to translate, some projects might be more attractive or useful than others. Therefore you should try to find a bit of information about a project to help you make your decision:

- How much translation work is required?
- How easy is it to submit a new translation?
- Does the software support any special needs of your language, such as right to left display for languages like Arabic, or special fonts?
- Are you able to use your preferred translation tools?
- Is this end-user software? Specialised software might not help you to have big impact.
- Does it run on multiple platforms like Windows and Linux? Are versions for all supported operating systems available at the same time?
- Will users be able to activate the localised interface easily, or will some configuration be required?
- Does the software require translations of other software like software libraries? If so, you should probably ask all these same questions about the other pieces of software.

Obviously you don't need to restrict your involvement to projects that are ideal in all regards. You might simply want to translate something because it is fun or because you personally use it. But any software that is harder than necessary might break your speed, and can be demotivating.

Partial localisation

While we would usually want to translate everything in a given piece of software, there are definitely some parts of the software that are more important than others to translate. If we can't translate everything, we can look for ways to take some shortcuts, especially for the first translation in our language. We should therefore look for ways of maximising our coverage while reducing the scope of the work.

In some cases, especially for very technical text, it might even be better to leave certain things untranslated. If things like error messages contain many new terms that are not well established yet, users who are very familiar with the English might be put off using the translated version. Remember that your audience will develop with you – the more people use localised software, the more they will get used it. If you are starting with new localisation, skipping things like the technical error messages in the internals of the software just makes sense, and provides you with an opportunity to work on things that more users will see. You will also get more satisfaction initially from more visible parts of the software.

An easy way to reduce the number of words, is to only translate certain files in a project with several files. If it is easy to identify the more important files, this could help you get started. If the project has only one file, or the important work is in all the files of the project, we might want to look for a way to extract the important things to work on. A tool from the Translate Toolkit called **pogrep** is a great help in this regard (see page 11). You could for example only extract short strings, or only strings that don't come from programming files with the word "admin" in them. Read more about pogrep here:

http://translate.sourceforge.net/wiki/toolkit/pogrep

Non-textual localisation

Although we discuss a lot of issues relating to translation, some projects might contain more than just translation work. A typing program might need information about the characters in your language and words for the typing exercises sorted by difficulty. Some games allow you to create sound files in your language. A word processor might allow you to specify a list of common typing mistakes to correct.

While you are translating you might notice some mistakes in the original text, or realise that it can't be translated properly because the programmers made certain assumptions about languages or cultures that aren't true in your case. Help the programmers by reporting these issues and thereby make translation easier for other people as well.

Discussing all of these issues are outside the scope of this book, but in this section we discuss some non-textual localisation that is a little more common.

Number and date formats

There are different traditions for writing numbers and dates around the world. Some languages have their own digits – an entirely different set of symbols to represent 0,1,2...9. Even in languages that use the same digits as English, there are different conventions about how to show fractions, and how to show large numbers. The same number and date is written differently in different places around the world.

Number example

```
10509 (unformatted – ten thousand five hundred and nine)
10,509 (USA)
10.509 (Germany)
൧൦,൫൦൯ (India – Malayalam)
```

Date example

```
15 March 2010 (2010-03-15)
March 15, 2010, 03/15/2010 (USA)
15. März 2010 (Germany)
```

Ensure that you are familiar with the correct custom for writing numbers and dates in your language. Some projects might ask you to specify how numbers should be formatted by the software. This is usually done as part of locales (refer to page 13), although it might be duplicated in a piece of software, and might even be indicated with special variables in the translation file. Refer to the section on date and time variables on page 49.

Currency and units of measurement

When software displays amounts of money (such as accounting software), it should not only format numbers correctly, but also indicate the correct currency for the relevant country. Similarly, countries might use different units to measure physical amounts such as distance, weight, etc. Always check that software follows the correct customs for your language or country. In some cases it might be necessary to convert the unit of measurement in translation. You will find some websites or software that can help you do a conversion from miles to kilometres, for example. However, when a physical quantity is only used as an example (usually with a nice round number), consider simply leaving it

at a round number and changing the unit. For example, when a game talks about some gold weighing 100 pounds, changing it to 50 kg might make it easier to read than 46.36 kg without impacting the message. Make sure that it is definitely safe to do this. If you are unsure, translate it more literally or convert the unit.

Images

Images are a good tool to help with understanding. However two people from different cultures can look at the same image and react differently or understand it differently. This has a lot to do with the background, culture and the environment an individual lives in. Images might make references to geographical features, activities, brand names, etc. that are not well known to the target audience, or which might not convey the intended meaning.

You might do localisation for years without ever needing to change an image, but it is good to be aware of the fact that images aren't necessarily suitable for your target audience. Always consider if an image might have a negative connotation, or if it might not succeed in communicating the correct message.

In some cases images might contain text which might need to be localised. This is usually bad practice by the programmers, but there might be valid reasons to do this.

Images can be edited in a number of different programs, but it is best to check with the programmers of the project which files should be used, and if the software supports alternate images for your localisation. You don't want to spend time working on images with no result. Take extra care to review any text you have to place in images – since you are probably not editing this in a translation program, it is easier for spelling mistakes or other inconsistencies to happen.

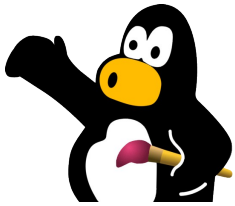
Sorting and lists

Many languages have their own rules to describe the desired order in which words are sorted alphabetically. Even languages that use the Latin alphabet might have specific rules on how to sort specific words or character patterns due to the nature of the language, or due to historical practices. Check for places where sorting might be relevant, and check that the correct rules are used for your language. This is usually obtained from a locale or a programming library.

Chapter 11: Project case studies

In this section, we show a few case studies of well known FOSS projects, and how you might want to evaluate them. You might want to refer back to the section on "Choosing the product or project" on page 74. We will be using some of the criteria mentioned in that section to evaluate these projects here.

Tux Paint



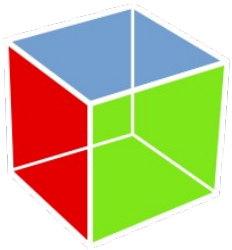
Tux Paint is a drawing program for kids.

- A translation of the application only is about 1500 words.
- It is trivial to submit new translations – an e-email to the programmer is enough.
- It has some support for advanced text layout, but each language would have to check for any special requirements. It supports right to left languages, several Asian languages, and languages needing complex text layout.
- Plain Gettext PO files are used, supported by Virtaal and a wide selection of translation tools.
- It is aimed at end users, and specifically children who might not know any second language.
- It is cross-platform software, available for many operating systems.
- A small change is required to test the localised interface. However, it is usually easy to replace an existing language's file to test. There is a configuration program to select the language as soon as it is one of the official languages.
- No other software needs to be translated, although the website and extra functionality can also be translated.

From the list it should be obvious that Tux Paint is a very attractive program for localisation. The text isn't very technical, although you might have to think a bit about some of the colours and shapes. It is an ideal project for someone who is new to localisation. It is fun, and the work pays off quite easily.

http://www.tuxpaint.org/

GTK+ and xdg-user-dirs



GTK+ is an underlying platform used for building a lot of software, but maybe best known for its use in the GNOME project. GTK+ provides software with several things that are common in software: dialogue boxes to open and save files, print options, and most importantly for this part of the book, some commonly used strings for common things like “OK”, “Cancel”, “Preferences”, “Copy”, “Paste”, etc. Since many programs use these strings from GTK+, it is useful to look at translating the strings from the GTK+ project for the benefit of all these programs. Although the translation file for GKT+ is a bit big, it is mostly quite easy to identify the most interesting strings in the file from sight, or by looking for the strings marked as “stock items”.

A project called xdg-user-dirs has a small translation file for the names of the common directories in each user’s account. “Desktop”, “Documents”, “Music”, “Pictures” – these directories are commonly seen when opening and saving files, so it provides another way to have an impact in many pieces of software without a lot of work.

Let’s repeat our evaluation based on our earlier questions:

- GTK+ contains many words, but the stock items are only about 100 words. Xdg-user-dirs is less than 50 words.
- GTK+ translations are submitted at the GNOME project, which is quite easy with several ways of submitting possible. Xdg-user-dirs is submitted with the “Translation Project” which is a bit harder, but you might also be able to ask someone else to help you with that.
- No special considerations should be necessary for your language.
- Plain Gettext PO files are used, supported by Virtaal and a wide selection of translation tools.
- GTK+ is used with software running on several platforms. Xdg-user-dirs is mostly used on Linux-based systems.
- These translations mostly rely on your system’s language selection.
- Xdg-user-dirs doesn’t dependent on external libraries. Small parts of GTK+ use translations from Glib, which won’t be very important initially. Glib can also be translated at the GNOME project.

These two projects present a very small amount of work to start having a positive impact in several applications that use them. The GTK+ translation will be especially effective for the GNOME desktop where most applications will use at least some of these strings.

Firefox

Firefox is a popular web browser from the Mozilla project.

- Firefox is a humongous project to translate, with over 20 thousand words for the application alone, and several other things necessary to become an official translation. Incomplete translations are not accepted.
- Submitting new translations to the Mozilla project is extraordinarily hard, with many technical steps, several of them involving encryption keys and version control systems. Only already complete translations are accepted.
- Firefox has good support for many languages, including complex text layout, and bundling of fonts for specific localised versions.
- The Translate Toolkit has converters that allow translating Mozilla software with PO files. However, these are not endorsed by the Mozilla project, and require translation teams to do it themselves. Other software is available, but nothing that translators would be used to from other projects.
- Firefox is end-user software, and very popular.
- It is cross-platform software, available for several operating systems.
- Official Firefox localisations have separate downloads. Alternatively language packs can be installed, but needs an additional add-on to activate the new interface language.
- No other software needs translation, but several product pages and marketing material need to be translated.

Firefox is very popular, and a good target for localisation in very experienced teams. The text is quite technical in places. It is a humongous amount of work with no reward while you are doing the translation. If you can make the investment, the rewards can be big, since Firefox is used by many people, and Mozilla marketing will in effect also help you to promote your work.

The Debian installer

The Debian project is a big FOSS project well known for their Linux distribution. It is also the basis used for building Ubuntu Linux. The installer could be the first software that people see when installing Debian or Ubuntu, and is a well managed project with a big focus on supporting languages.

- It is not required to translate the complete installer, but at least two files need to be at 100%. This is more than 10 thousand words.
- The project is very welcoming and accepts contributions in a few different ways, among them through their Pootle server.
- The Debian installer has good support for many languages, including complex text layout, and bundling of fonts for specific localised versions.
- Plain Gettext PO files are used, supported by Virtaal and a wide selection of translation tools. They also host a Pootle server.
- The installer is not really end-user software, and more likely to be used by technically inclined people. The installer is used infrequently.
- This software is only relevant to certain Linux distributions.
- Once you are an official language, the users will select the language easily from a list. Testing might be a bit harder.
- The translation of some country names are necessary, and several other files are available to translate the installer more completely. These are not strict requirements, however.

The Debian installer might be a nice target for technically inclined translators, but it will not reach many users, and the text is quite technical in places. The requirements for entering are quite high, and involves translating a lot of strings that don't have high impact (such as time zones). The project is well managed, and values translations. With a well-resourced team, you might want to consider this eventually.