Internationalisation and localisation in Debian

Javier Fernández-Sanguino Peña
The Debian Project

jfs@debian.org

Christian Perrier
The Debian Project

bubulle@debian.org

Internationalisation and localisation in Debian

by Javier Fernández-Sanguino Peña

by Christian Perrier

Debian GNU/Linux is one of the most ambitious free software project, putting together a large number of developers from all around the world working in building a free operating system.

This document describes the rationale and management of internationalisation and localisation within the Debian project, including the current infrastructure and tools available to both translators and package maintainers.

Keywords: free software, translation, internationalisation, localisation, operating system

Table of Contents

Chapter 1. Introduction

The following document describes the approach taken for handling internationalisation and localisation in the Debian project describing the infrastructure, and tools available for translators and package maintainers as well as the work done by the different translation team members.

1.1. English as the official language?

English is often regarded as the official language of most free software projects. This assumption affects users of free software and users of information systems generally, to the degree that "They should all just speak English!" is a frequent reaction to internationalisation projects. On behalf of the majority of the world's population - who do not speak English - a great deal of work is done voluntarily in the Debian project, to ensure that program interfaces do not only present their messages in English and that most documentation is not only written in English.

Currently, most of the working groups within Debian use English as their main language for information exchange (discussions, documentation, etc.). Developers, even if not native speakers, program code with an English interface and write documentation in English. The main information sources of official documentation (including the project website) are first written in English.

A single working language helps coordinate work of multiple developers worldwide, and helps people exchange their ideas. Englis is the common language for communication with Debian developers (http://www.debian.org/contact.en.html). However, using only English disadvantages many potential users and limits the pool of potential developers. It's an accessibility issue, as well as a usability issue. We want everyone to be able to use their computer effectively, regardless of the languages they are familiar with or can understand, and that's what internationalisation is about.

1.2. The importance of internationalisation and localisation

Debian's goal of providing a *universal* operating system is based on the voluntary work of many translation teams. Having a *universal* operating system does not only mean having all possible software tools available, it also means providing an operating system that can be used by anyone worldwide. Even though English is a very widely-used language in the IT world today, this does not mean that all the world population can speak it, or even understand it. As we have pointed out, in fact, the majority of the world's population cannot use English. That is why this goal is so important and why we need to fulfill it, at least partly, by providing an operating system accessible to every user, in a language they are familiar with, regardless of what that language is. The more languages we support, the more people we reach. It's as simple as that.

This means Debian needs to provide an installation system that can also be used by non-English speakers, and that can set up a system also suitable for use for non-English speakers. Debian users need to be able to read about the operating system itself, itself, so documentation must be translated, as well as application interfaces. Both the documentation available in the operating system itself, and the documentation available through other sources (e.g. the project's website). Debian users need to be able to ask for help, so mailing lists, IRC channels, fora and other interactive help systems need be available to them in a language they can use.

Making Debian usable by more people is also a selfish goal for the project: having more users means that the software developed by the project gets used more (more exposure leads to more testing and eventually, more features) and it also means that the new users might eventually contribute to the project (through bug reports, patches, translations) and even become Debian developers themselves. More access simply means more resources.

1.3. The internationalisation and localisation process

According to Introduction to i18n (http://www.debian.org/doc/manuals/intro-i18n) from Tomohiro KUBOTA: "I18N (internationalisation) means modification of a software or related technologies so that a software can potentially handle multiple languages, customs, and so on in the world." while "L10N (localisation) means implementation of a specific language for an already internationalised software."

Internationalisation is the process that makes a program capable of providing a computing environment for the user adapted to his/her own language, currency, date and time formats, etc. Most users of a similar background will share an environment, and this means there a number of common environments that a program has to support. The term internationalisation is often abbreviated to i18n.

Even if a piece of software is ready to use different environments, that does not mean that it can immediately do so, since specialists (typically people who use that environment) need to adapt it so that it can be used in a language a user is familiar with. This process of adapting the software to a specific environment is called localisation. Localisation is typically focused on the translation of each and every message that the program can handle (and implements in the user interface, such as menus, buttons, error messages and tooltips). Localisation is typically abbreviated to l10n.

L10n and I18n are closely related, but the issues in achieving each of them are very different. It's not really difficult to allow the program to change the language in which texts are displayed, based on user settings, but it is very time consuming indeed, actually to translate all its messages. On the other hand, setting the character encoding may be trivial, but adapting the code to use several character encodings can be a challenging and complex task.

This document will not try to explain all the different issues facing localisation and associated with the representation of different code pages. However, it's important to recognize that this is a key issue, especially for environments that do not use the Western code pages (ASCII character set and ISO-8859), such as Asian and Indic scripts and languages with combined diacritics. We need to bear in mind that the

majority of potential users world-wide use languages which are not covered by those two very limited code pages. UTF-8 support is a crucial step in internationalization.

1.4. Internationalising, translating and being internationalised and translated

Debian supports an ever-increasing number of languages. Even if you are a native English speaker and do not speak any other language, it is part of your duty as a maintainer to be aware of issues of internationalisation. Internationalisation is the bridge between you and most of your potential users. Therefore, even if you yourself can function effectively in English, this chapter gives you information you need as a maintainer.

Like the overall i18n implementation, which still has to coalesce into a standard process, l10n within Debian does not yet have a central infrastructure which could be compared to the dbuild mechanism for porting. Currently, most of the work has to be done manually.

1.4.1. How are translations managed within Debian?

Handling translation of the texts contained in a package is still a manual task, and the process depends on the kind of text you want to see translated.

For program messages, the gettext infrastructure is in common usage. For applications the translation is handled upstream, within projects like the Translation Project (http://translation.sourceforge.net/), the Gnome Translation Project (http://developer.gnome.org/projects/gtp/) or the KDE Translation Project (http://i18n.kde.org/). The only centralised l10n resource within Debian is the overall Debian translation statistics (http://www.debian.org/intl/l10n/), where you can find information about the translation status of files found in an actual package. This is similar to Fedora's Translation Status page (http://i18n.redhat.com/cgi-bin/i18n-status), but unfortunately different, at this stage, in that we don't yet have a common (unlike it) there is no common infrastructure to facilitate the translation process.

An effort to translate the package descriptions started quite some time ago, even though very few tools currently support these translated descriptions (i.e. only APT can use them, when configured correctly). Maintainers do not need to do anything for this to work, and the translators should use the DDTP. For more information see Section 2.2.2.

For debconf templates, maintainers should use the po-debconf package to help translators work more effectively. Some statistics of the integration of debconf translations in packages can be found on the overall Debian translation statistics (http://www.debian.org/intl/l10n/) pages. For more information, see Section 2.2.1.1

For web pages, each l10n team has access to the appropiate CVS, and the statistics are available at the overall Debian translation statistics site. For more information see Section 2.1.

For general documentation about Debian, the process is more or less the same as for the web pages (the translators typically need access to the CVS or SVN repository holding the documentation files), but there is no statistics pages yet, except for a few specific projects. Central information is definitely needed. For more information see Section 2.4.

For package-specific documentation (man pages, info documents, and information in other formats, like XML/Docbook, HTML and PDF), almost everything has yet to be done. The KDE and GNOME projects handles translation of their documentation in the same way as their program messages, but there is no specific way to handle translation of documentation in Debian project-wide[1]. The translation of Debian-specific man pages was initially handled within the manpages module (http://cvs.debian.org/manpages/?cvsroot=debian-doc) of the DDP CVS repository, but in some packages, manpages are handled just like generic documentation. For more information on how manpages translations are handled, see Section 2.4.2.

1.4.2. I18N & L10N FAQ for maintainers

This is a list of issues that Debian package maintainers may face concerning I18n and L10n. While reading this, keep in mind that there is not yet any real consensus on those points within Debian, and that they are simply helpful advice. If you have a better solution to a given issue, or if you disagree on some points, feel free to provide your feedback, so that this document can be enhanced.

How do I get a given text translated?

> For translate package description or debconf templates, you have do not need to do anything at all. The DDTP infrastructure will send the translatable descriptions volunteers, and process the resulting translations with no need for you to interact. Addition of debconf templates is followed by translation teams that will send you new translations for them through the Bug Tracking System (you can contact debian-i18n, however, if you would like translations before releasing a new package version).

> For all other material (PO files for applications, man pages or other documentation), the best solution is to make your text available somewhere on the Internet, and ask the translators on the debian-i18n mailing list to translate your file(s) into their different languages. The translation team leaders, at least, are subscribed to this list, and they will take care of the translation and of the quality-assurance processes. Once this is complete, your translated material will be emailed to you. You may receive queries about context for specific points (inserting context in your document will always ensure a better quality of translation), and get typo reports as a bonus feature.

How do I get a given translation reviewed?

> From time to time, somebody might translate some texts included in your package and will ask you to include it when you release. Understandably, you want to know if this translation is up to

standard. You don't want to be releasing a campaign speech or a dirty limerick instead. Therefore, you can send the document to the Debian l10n mailing list for that language, asking for a review. Once that is complete, you can feel more confident in the quality of that translation, and can include it in your package with pride. You may also have introduced a new translator to that language team: a valuable resource for Debian.

How do I get a given translation updated?

If you already have some translations of a given text, each time you update the original, you should also politely ask the previous translator to update his/her work, to bring that translation up to date with regard to the current original text. Keep in mind that this task takes time, just as it took you time to update the original: allow for at least one week to get your translation updated, reviewed and returned.

If the translator doesn't respond for some reason, please contact the Debian l10n mailling list for that language. The team-leader should respond promptly. If you don't hear back from the team, please email the Debian i18n mailing list, so it can follow this up or provide alternate contacts for the team. If, somehow, the translation doesn't get done in time (and this is rare, when you follow these procedures), don't forget to put a warning [2] in the translated document, stating that the translation is outdated, so the reader should also refer to the original document if possible.

You should avoid removing a translation completely, simply because it is outdated. An old document is usually better than no document at all for many users. Old translations might also include glossary information that might be useful for future reviews and you never know when a new translator will use the old translation to provide an updated translation (which is typically faster than writting a new translation from scratch).

How do I handle a bug report concerning a translation?

These reports should really go to the translation team, whose contact details are shown in all PO file headers, and should be shown on all documentation. It's a good idea to advise users to send any i18n bug reports straight to language teams[3]. When you do get them, please mark the bug as "forwarded to upstream", and forward it both to the previous translator and to his/her language team (using the corresponding debian-l10n-XXX mailing list).

How can I (as a maintainer) help the translation effort?

You can eliminate a lot of wasted effort, and confusion, by organizing your end of the l10n process. Choose carefully what you want to translate (Debconf messages are always translated, but you might want to have additional items translated), and integrate translation with your release schedule (of either packages or new upstream versions). Allow for a string freeze before a release or package upload: this gives translators time to complete and review translations. Plan to request translations on the debian-i18n mailing list, at a realistic point before release, then again when you announce the string freeze. Communicate readily with debian-i18n and your translation teams: keep the information flowing.

If you are the maintainer of a popular package which might get translations after a release, you might get updated translations for upstream content (program messages and documentation) as bug

reports or see changes in upstream's revision control system after you have uploaded a new version. Instead of sitting on the translations waiting for upstream to add them and making a release, you can make this translations available in the Debian package inmediately. This ensures that Debian users will see the translations as soon as they are available and it also boosts the morale of translators which see that their work is made available to users. Translators might get frustated if they send you updated translations for a package (even after sending them to upstream or updated upstream's repository) and do not see them being included in the Debian package until upstream decides to make a new upstream release. Unlike patches for source code, which might introduce regressions and instability, changes to PO files or new translated manpages do not affect the program itself and can be safely added.

1.4.3. I18N & L10N FAQ for translators

While reading this, please keep in mind that there is no general procedure within Debian concerning those points, yet, and that in any case, the most effective way to resolve any issue is to work with your language teams. Language teams are set up to ensure an integrated approach to translation for that language, and that integration includes interfacing with developers.

How can I help the translation effort?

If you are able to join the translation effort, welcome! We appreciate your effort. As a quick guide: decide what you want to translate, make sure that nobody is already working on it (using your debian-l10n-XXX mailing list), translate it, get it reviewed by other native speaker on your l10n mailing list, and provide it to the maintainer of the package (see next point). The Debian i18n web pages, especially the Translation HowTo, can give you more information.

How can I provide a translation for inclusion in a package?

Most importantly, make sure that your translation is correct (by asking for review on your l10n mailing list) before providing it for inclusion. It will save time for everyone, and avoid the chaos resulting in having several versions of the same document in different bug reports. Translation teams are capable of quality work, and that is the reputation we want for Debian i18n.

The recommended way to submit translations is to file a regular bug against the package, with the translation file attached. Make sure to use the 'PATCH' tag, and to not use a gravity higher than 'wishlist', since lack of translations doesn't actually prevent a program from running[4]. For more information please read Section 4.6.

1.4.4. Best current practice concerning l10n

- As a maintainer, never edit the translations in any way (even to reformat the layout) without asking on the corresponding l10n mailing list. You risk, for example, breaking the encoding of the file by doing

so. Moreover, what you consider as an error can be right (or even needed) in the given language. Trust the expertise of translators these areas.

• As a translator, if you find an error in the original text, please report it (by sending in another bug report, also of "wishlist" priority). Often, translators are the most attentive readers of a given text, and if they don't report the errors they find, nobody will. Also, it makes your job easier, if the original text is correct.

• In any case, remember that the major issue with l10n is that it requires several people to cooperate effectively, and that it is all too easy to start a flamewar about small problems, because of a misunderstanding. Avoid misunderstandings by being as clear as you can, using straight-forward language (avoid idiom, for example) and not making assumptions. If you do run into difficulties, please ask for help on the corresponding l10n mailing list, on debian-i18n, or even on debian-devel.

• Essentially, cooperation can only be achieved through *mutual respect*. We are all working hard in the same project: we achieve so much more when we work together.

Notes

1. The Translation Project and GNU are curently running a pilot project on the manpage translation (and distribution) process which we may be able to implement.

2. If the translation is managed through gettext you do not need to do this as the gettext tools will prevent the user from seeing translations that have not been updated.

3. Upstream users can add a link on their site, to the Debian language teams listing to prevent getting these mails

4. It might, however, prevent people from using it

Chapter 2. I18n/L10n projects in Debian

Debian has different ongoing processes, to facilitate the internationalisation of the software it develops, as well as to assist translators in working on the localisation of content produced for Debian. This chapter explains the different i18n and l10n ongoing projects in Debian, and describes how they work.

2.1. Translation of the Debian website

A project website is so often used as the primary information source for both the users of a program, and for people who want to learn about it but don't use it yet. Information on a project's website is usually complementary to the documentation available through other means. This website information is also typically more up-to-date than the documentation provided when the software was installed, or the distribution media in which it provided was purchased.

This gives the website some priority in the internationalisation and localisation process of a universal project. It's out there, it's current, and it's being viewed: so it is translate, to make it accessible to many more people. A website is a moving target for translation, requiring frequent updates, but it's a key access point for users (both existing and future).

In order to present translated information to the user, the Debian project website uses *Content Negotiation* technology (more specifically, Apache's content negotiation (http://www.apache.org/docs/content-negotiation.html)) that is based on the user's web browser providing information on chosen languages (*Preferred-Language* HTTP header). The user may set this language preference individually in the browser, or simply ask the browser to follow the system preferences. Content negotiation, in this case, means that the website software can present a copy of the content translated to the user's most effective language (if a translated copy is available). More information on content negotiation is available on the Debian website's description of content negotiation (http://www.debian.org/intro/cn). Translated websites will typically include a "language bar" which shows in which languages that site is available.

In this way, server administrators will manage both the original document (typically in English) and multiple translations of that document. Content negotiation takes care of the task of looking for the document for a given URL in the nominated language, either showing it if it is available, or defaulting back to the original page.

Translations for the Debian website are being created for 33 world languages, although only 10 language temas have more than 500 pages translated out of over 3000 available pages, and only 5 teams have translated more than 50% of the site. It is a very large and demanding task, but one we assign priority, so there is a great deal of translation activity around the Debian website. New languages will continue to be added, and further pages translated.

The web server is based on content managed by the **wml** program ,which allows separation of templates

and content, and also provides a mechanism to generate content within the pages on *compile*. Wml is to HTML what a C source is to its object code.

Most of the information on the web server is available in a directory tree, handled through CVS (at cvs.debian.org). In that tree, there is a directory for every language in which the website is translated. In principle, all the languages follow the directory hierarchy defined in the original (English) pages although, depending on the actual content translated, the contents of each subdirectory within a language might vary. There is also a specific location for translation teams' content which is not translations, but rather specific content written in that language, usually specific to that language/culture. The mechanism for generating the pages is based on **Makefile** in such a way that all the wml files in a given directory are compiled and published together. The use of included files means translators do not need to concern themselves with the content of the Makefile files.

Thanks to the independence of real content and aesthetic information, translators can simply take the original (English) wml files, move them to their own directories and translate them. The templates that generate the website itself need not be translated, unless specifically required. The templates use wml's internationalisation tools and **gettext** to extract and present the translatable information (including headers, footers, menus, buttons, both interactive and passive text) in all the pages for translation.

One of the main issues in ongoing translation is the need to be able to monitor changes in the original content. In particular, translators need to know which changes require updating translations. The web server has the same problem. Using content negotiation to detect a user's language settings, is only looking to see if a translation exists and can be displayed, and is not aware of the status of any translated content. So it's possible that users might be presented with out-of-date information. This issue would not exist if the translation were removed when it was no longer current, since the web server would then simply provide the default language (English) to the user. However, many changes to the website are not fundamental changes, or are typo fixes to the original document, so it would seem senseless to provide the original document in these situations, since the translation (even if out of date) would still be useful to website readers who can't understand the original content.

To deal with this currency issue proactively, the website development team implemented a mechanism to detect out-of-date translations, based on *translation headers*. These headers are included in translated documents, describing which original file they translated, and specifying the CVS revision used. The use of this header makes it possible to introduce several additional tools to facilitate effective translations:

• an automatic program that can be run in the directory tree to provide a translator with a list of the documents that have not been updated for a given language. This is done through comparison of the current CVS revision of the original file with the CVS revision that the translated document states it used as a base for the translation;

• a wml toolkit that, at the same time pages are compiled (once a day), checks if the wml file being handled is a translation or not, and if it is, if it is current. It uses this information to introduce a predefined text in the output document, that tells the reader if the translation is not up to date, and points to the original translation (the text varies based on how out-of-date the translation is). The same tools will also add a footer to all pages, to list the translations available not simply for the website in any degree, but for each specific page.

More information is available on the website development reference
(http://www.debian.org/devel/website/translating).

There are further tools, also based on this mechanism, that provide statistics related to the translation
effort (http://www.debian.org/devel/website/stats/), including statistics on out-of-date translations
(http://people.debian.org/~peterk/outdated/)[1].

Since translations have to follow the creation of the original content, they will typically lag slightly
behind the translated content in an active website. This mechanism provides a way for translators to
work on translations at their own pace, while ensuring users reading the content know how current it is,
and have the option to switch to the original content.

2.2. Translation of Debian tools

Internationalisation of the Debian distribution also involves the internationalisation and translation of the
tools developed within the distribution itself. This includes the translation of the installation system (**d-i**),
of the package management system tools (**dpkg**, **dselect**, **apt**, **aptitude**), of desktop menu management
(**menu**) and of other Debian tools such as **debconf**.

Without proper internationalisation or translation of these tools, the majority of current and potential
users will not be able to use or manage the Debian GNU/Linux operating system effectively.

The following sections describe the efforts in internationalisation and in the translation of different parts
of the Debian GNU/Linux distribution within the Debian project itself

2.2.1. Debconf translation

Debconf translation covers the translation of all interactions with the system administrator while
installing packages (or installing the entire system).

2.2.1.1. Maintainers' tasks - internationalisation

The Developer's reference (http://www.debian.org/doc/manuals/developers-reference/) recommends
using the debconf protocol for user interaction. This presents information in a standardised way, easier
for the user to follow and understand, and also allows more effective localisation of user interaction.
Allowing debconf templates to be translatable is also a recommended practice
(http://www.debian.org/doc/manuals/developers-reference/ch-best-pkging-practices.en.html#s-bpp-
i18n).

This requirement could even become mandatory in the future. This was mentioned for Etch but unfortunately no-one followed through, by proposing the required changes to the policy for this to take place. Debconf, untranslated, is a bottleneck between the users and your software. Translate it, and it's an access conduit.

Debconf translations involve the **po-debconf** package which has now become a standard in Debian. Even though not mandated by the policy, all packages using **debconf** for user interaction should use **po-debconf**. It automates a number of the repetitive but essential tasks, and ensures that translators receive update notices for your templates. This is best practice in the field, ensuring a much higher translation rate.

The details of **po-debconf** are well covered in its man page (**po-debconf(7)**). In short, strings (Description, Choices and Default fields) should be prepended with the underscore character in the debconf templates file. If you do so, they will be included automatically in the list of translatable material.

The **debconf-updatepo** program gathers all the translatable material in a Portable Object Template (POT) file located under `debian/po` in the package build tree. This templates file is then used as a reference by translators to create a Portable Object file for their languages, using the language ISO 639 (http://www.loc.gov/standards/iso639-2/englangn.html) code.

2.2.1.1.1. Consistency of style

Users will often see a series of several debconf screens in a row. A consistent style in both writing and presentation considerably improves the perception of professionalism in the overall Debian distribution.

Unfortunately, the original English strings very often lack this consistency, and despite some efforts in the Developer's Reference (http://www.debian.org/doc/developers-reference/ch-best-pkging-practices.fr.html#s-bpp-config-mgmt) to give maintainers advice about writing style for debconf templates, the manner of addressing users often varies wildly:

- use of the first person;
- use or not of interrogative form for input-style templates;
- repeated interrogative form in long and short description of templates;
- varying enumeration styles;
- use of specific references to some of the debconf interface-specific widgets or behaviour (talking about "previous" or "next" screens, assuming that boolean templates use Yes/No style questions, etc.).

In general, maintainers should follow translators' advice when they suggest improvements to original templates.

The French team, and more specifically Thomas Huriaux, has setup a dedicated page (http://haydn.debian.org/~thuriaux-guest/templates/templates_by_packages.html) which automatically tracks down the most common "errors" with regards to this recommendation.

2.2.1.1.2. Repetitive strings

A great deal of user interaction in debconf templates involves similar questions or input, such as web server configuration for packages that provide web services, database interactions for RDBMS-related packages or packages that use databases, basic identification, etc.

To avoid such repetition and the wasted time associated with it, packages aimed at providing common methods, as well as common debconf templates for such use have recently appeared in the Debian distribution. The **dbconfig-common** package is one of them.

Maintainers are encouraged to use these packages, and to help improve them for better quality and efficiency. The use of the debconf *register* commands is also encouraged in some cases, when a package needs to use strings or templates provided by another package.

2.2.1.1.3. Translation maintenance - localisation assistants

The part of this paper which deals with i18n/l10n of Debian specific programs(see Section 2.2) introduces the concept of "localisation assistants".

As the maintenance of debconf translations is usually very simple, this paper does not enforce the use of localisation assistants, except in the case of packages with a high number of templates and/or packages listed a top priority packages for translators (packages which belong to the Debian Installer "levels" packages, or very popular packages).

However, in any situation where the maintenance of debconf translations becomes a hassle for them, package maintainers are encouraged to cooperate with a localisation assistant. The debian-i18n mailing list (http://lists.debian.org/debian-i18n) is the entry point that should be used to "recruit" localisation assistants.

2.2.1.2. Translation work - localisation

2.2.1.2.1. Getting translation material

Although everyone can access debconf translation material by downloading each package source and looking in `debian/po`, translators are encouraged to use links from the translation statistics pages (http://www.debian.org/intl/l10n/po-debconf/).

2.2.1.2.2. Priorities

Although debconf templates are fairly small, there are over 695 individual debconf templates, a fairly large and repetitive task for translators. The number of Debian packages with translatable material for debconf is pretty important. As of this writing these 695 packages with translatable debconf material add up to about 10000 strings to translate.

Recent changes to the translation statistics pages now allow sorting packages by their popularity, using data from the **popularity-contest** package. This was achieved to avoid translators spending their effective time on rarely-used packages, and to ensure commonly-used packages get translated first.

2.2.1.2.3. Statistics

Statistics for debconf translations are gathered on the Debian l10n status pages (http://www.debian.org/intl/l10n/po-debconf/rank) and are separated from the translation statistics for individual programs. These pages also give access to the translation material collected by the statistics robot operated by the Debian i18n team.

2.2.1.2.4. Translation

Translation usually involves copying the `templates.pot` file to `<code>.po` (where *code* is the language code it is being translated to), filling in this file's header with appropriate information, then using a Portable object editing tool (or any text editor) to enter the translations for all the strings.

2.2.1.2.5. Checking translations

Translators can test the debconf PO files by using the **podebconf-display-po** tool from the **po-debconf** package. They will need to install this package on their system.

A locale for the tested language must be built on the system (which can be achieved by "dpkg-reconfigure locales") and the LC_MESSAGES variable should be set to this locale.

It is recommended to test the debconf templates by using the debconf dialog interface in a 80x25 screen.

podebconf-display-po has a few quirks, especially when some strings are shared among several templates, so it should not be relied on blindly. Despite this, it has already proven extremely helpful in detecting formatting issues.

2.2.1.2.6. Style consistency

Translations need a consistent writing style, just as the original strings do, to communicate effectively. This information, and the way it is presented, also functions as the public face of that application or document, and of the overall project.

While it is true that, as yet, the original English strings in many packages may lack this professional consistency, translators can greatly help package maintainers a great deal in improving accuracy and style. They can report syntax and style errors as bug reports against the relevant packages. Ultimately, this partnership between package maintainers and translators not only produces an application accessible by many more people, but also a higher overall quality of presentation.

Translators should aim for consistency in their own translation work. Even if the original strings do not follow writing-style recommendations, translators should adapt their translation to be consistent in their language, to create a high-quality translation as an achievement in its own right.

Translation teams, and the QA work they do, have indeed a great role to play in this overall improvement of presentation consistency. They are capable not only of processing information for translation, but also of reviewing it as information. Through this additional review process, the quality of information presentation in debconf templates should continue to improve.

2.2.1.2.7. Sending new translations and translation updates to maintainers

Translators should send debconf translation updates to maintainers via the Bug Tracking System (http://bugs.debian.org/), even when the update is requested by a maintainer using an automated tool to call for updates. Such bug reports should use the "l10n" and "patch" tags, and severity "wishlist"

They should be filed against the *source* package rather than against a binary package.

The use of the reportbug utility is recommended.

A standardised bug title is recommended: "<package>: [intl:<code>] <Language> debconf templates translation", where <package> is the source package name, <code> the language ISO code and <Language> is the language name in English.

Here's an example of reporting the French translation of the **geneweb** package. Of course, in the example below, **geneweb** should be replaced by the appropriate package name:

```
bubulle@mykerinos:~/tmp> reportbug geneweb
*** Welcome to reportbug.  Use ? for help at prompts. ***
Using 'Christian Perrier <bubulle@debian.org>' as your from address.
Detected character set: UTF-8
Please change your locale if this is incorrect.
```

```
Getting status for geneweb...
Checking for newer versions at packages.debian.org...
Will send report to Debian (per lsb_release).
Querying Debian BTS for reports on geneweb (source)...
14 bug reports found:

(snip all bug reports for geneweb)

(1-14/14) Is the bug you found listed above [y|N|m|r|q|s|f|?]? n
Maintainer for geneweb is 'Christian Perrier <bubulle@debian.org>'.
Looking up dependencies of geneweb...
*** The following debconf settings were detected:
  geneweb/run_mode: Always on
  geneweb/remainingdir:
* geneweb/remove_databases: false
  geneweb/port: 2317
* geneweb/lang: French
Include these settings in your report [Y|n|q|?]? n

Please briefly describe your problem (you can elaborate in a moment; an empty response will
should be a concise summary of what is wrong with the package, for example, "fails to send
start with -q option specified."
> [INTL:fr] French debconf templates translation
Rewriting subject to 'geneweb: [INTL:fr] French debconf templates translation'

Enter any additional addresses this report should be sent to; press ENTER after each addres
blank line to continue.
>

How would you rate the severity of this problem or report?

1 critical       makes unrelated software on the system (or the whole system) break, or ca
                 loss, or introduces a security hole on systems where you install the pack
2 grave          makes the package in question unusable by most or all users, or causes da
                 a security hole allowing access to the accounts of users who use the pack
3 serious        is a severe violation of Debian policy (that is, the problem is a violati
                 'required' directive); may or may not affect the usability of the package
                 policy violations may be 'normal,' 'minor,' or 'wishlist' bugs. (Package
                 designate other bugs as 'serious' and thus release-critical; however, end
                 so.)
4 important      a bug which has a major effect on the usability of a package, without ren
                 unusable to everyone.
5 does-not-build a bug that stops the package from being built from source. (This is a 'vi
6 normal         a bug that does not undermine the usability of the whole package; for exa
                 particular option or menu item.
7 minor          things like spelling mistakes and other minor cosmetic errors that do not
                 functionality of the package.
8 wishlist       suggestions and requests for new features.

Please select a severity level: [normal] 8
Do any of the following apply to this report?
```

```
1 l10n          This bug reports a localization/internationalization issue.

2 patch         You are including a patch to fix this problem.

3 experimental  This bug only applies to a package in the experimental branch of Debian.

4 none

Please select tags: (one at a time) [none] 1
- selected: l10n
Please select tags: (one at a time) [done] 2
- selected: l10n patch
Please select tags: (one at a time) [done]

(snip editor being spawned)


Report will be sent to "Debian Bug Tracking System" <submit@bugs.debian.org>
Submit this report on geneweb (e to edit) [Y|n|a|c|e|i|l|m|p|q|?]? a
Choose a file to attach: /home/bubulle/tmp/fr.po
Report will be sent to "Debian Bug Tracking System" <submit@bugs.debian.org>
Attachments:
 /home/bubulle/tmp/fr.po
Submit this report on geneweb (e to edit) [Y|n|a|c|e|i|l|m|p|q|?]? y
```

2.2.2. DDTP

Another translation project is the translation of the descriptions of the software packages offered for the Debian GNU/Linux OS. This project is named the *Debian Description Translation Project* (DDTP).

Each package shipped in Debian GNU/Linux is provided with two descriptions: a short description (less than 80 characters) and a long one (of variable length). The first one describes briefly what the package does and the second one describes its main functionalities, characteristics, differences with other software, etc. This information is vital for users who are looking for a given functionality within the (huge) quantity of software provided in the Debian distribution. The package management tools include interfaces to do word (or regular expression) searches through these descriptions.

However, the fact that all these descriptions are only available in the English language makes it difficult for users that have already installed the system to look for new software they might want. Users not fluent in English might not be capable to define their needs in a foreign language to look for the specific software they are looking for.

The description translation project started in order to solve this problem for end users. This project was started initially by two different work groups in the year 2002, resulting in two different management systems (one based on e-mail and another based on a web application[2]. The framework was based on

Perl scripts that collected descriptions and provided an e-mail interface for translators (translators could request new translation through email and submit them using the same interface). All translations are stored in a simple file with a database holding the status of the translation (new, needs to be reviewed, out of date, etc.). Most notably, the framework did not use PO files for translations which got it many objections from some translation teams.

The translations from the web based interface one of these systems have later been merged on with the other one. The project was used by different translation teams. When Debian systems were compromised (http://www.debian.org/News/2003/20031121) in November 2003 the main framework was disconnected pending a review of the source code.

The Brazilian team (Otavio Salvador) later developed patches for **apt** to support the use of the translated descriptions in it. This patch (apt-ddtp (http://people.ubuntu.com/~mvo/arch/ubuntu/apt--ddtp--0/)) was ported to the 0.6 code base, with the help of some other developers, including Michael Vogt. Later, in April 2005, an Alioth project (http://alioth.debian.org/projects/ddtp/) was started and the original source code of the translation framework was moved to a SVN repository (http://svn.debian.org/wsvn/ddtp/).

As of this writing the DDTP project does not have an interface to submit translations. The translations already done for descriptions can be downloaded from http://ddtp.debian.org/, but they are not being updated.

This work, however, has been reused in Ubuntu. Currently, the Ubuntu package descriptions are imported into Ubuntu's translation framework: Rosetta (http://launchpad.ubuntu.com/rosetta) using the po/pot extraction/assemble tools developed by Michael Vogt, and available at michael.vogt@ubuntu.com--2005/apt-ddtp-tools--main--0 (http://people.ubuntu.com/~mvo/arch/ubuntu/apt-ddtp-tools--main--0/). The tools converted the Packages file to a pot file that is then imported into Rosetta. Rosetta then generates the PO files from the translations that can be downloaded and converted into a Translations-$lang file that apt-ddtp understands.

2.3. Translation of installed systems

A common need for users installing a Debian system is to end up with a fully localised system that does not need to be set up in order to be useful for users who require a given locale for being supported. Besides having a fully I18n Debian Installer there are several things that help set up an internationalised Debian system.

2.3.1. Localization-config

Localization-config is a tool to setup the configuration of programs in order to use a default language for the system as defined by the system administrator. This tool was initially written by Konstantinos

Margaritis, developed for Skolelinux and was later integrated with Debian and uploaded in August 2004. It was later integrated in the Debian installer for sarge (through **base-config**).

This tool tries to ease the way that administrators (or, even, the Debian Installer) set up a fully localised system. In a system, localisation begins by defining the preferred locale but there are many tools that need to be specifically configured to use a given locale. This tool can run after and before package installation to adjust its configuration to use a given language through the use of a collection of scripts. Each of the available scripts adjust the configuration of a single package for a number of languages. Please note that this tool is targeted for system configuration. Users wishing to change their own locale (but not the system's) should use **set-language-env** (in the **language-env** package).

In order to use this tool, the system administrator have to call one script, **update-locale-config**, providing the desired locale as a parameter. This script calls all other scripts in `/usr/lib/localization-config`. If the -p flag is used, the *.preinst scripts are called, else (default) the *.postinst scripts are called. The preinst scripts are supposed to be run when you need to do configuration of a package before it is installed (main use is by preseeding debconf values of the package). The postinst scripts are there when a package does not support this kind of preconfiguration (such KDE, gdm, etc.) and has to be configured by modifying its configuration file.

In order to prevent modifying new configuration file formats the scripts first checks the version of the package it should configure, in three different ways:

- check the version of the installed package.

- check the version of the package available available in the configured repository (that is, what is the version of the package you would install if you use **apt**).

- if both of the checks above fail, try getting the version from `/etc/debian_version`. This method should never be called if everything works as expected.

The version of the package is used to define version maps, since some versions might need to be configured in slightly different ways. The wrapper scripts will call the appropriate script under the proper folder (woody, sarge, etc) and do the actual configuration.

Localization-config currently configures:

- Dictionaries-common (**ispell**): preseeds the preferred dictionary;

- fontconfig: implement fonts substitution (currently only for Greek);

- Gdm: sets the default language for sessions;

- KDE: modifies the language settings of `/etc/kde3/system.kdeglobals` and `/etc/kde3/kdm/kdmrc`. It can also modify the settings of the **ktouch** program by adjusting `/etc/kde3/ktouchrc`;

- Links: Sets the preferred language for browsing by modifying `/etc/links.cfg`;

- Ltsp: Sets the X keyboard map by modifying `/opt/ltsp/i386/etc/lts.conf`;

- Lynx: Sets the preferred language for browsing by modifying `/etc/lynx.cfg`;

- Mozilla: Sets the preferred language for browsing by setting the UserAgent locale and accepted languages at `/etc/mozilla/prefs.js`;
- Xfree86 and Xorg: preseeds the character set for the language.

Localization-config is still work in progress and members of the different i18n teams still need to review the available scripts in order to integrate their own languages in the installation process. Moreover, due to the changes introduced in the Debian Installer early in 2006 (in which base-config was removed and introduced into the first stage of the installed), this package needs to be re-integrated into the *etch* installer.

New i18n scripts need to be written also for programs such as: **locale-purge** (`/etc/locale.nopurge`, TeX/LaTeX (including hyphenation rules defined at `/etc/texmf/language.dat`), console settings (including `/etc/inputrc` and the console keymap by preseeding **console-common**), Mutt (locale and charset at `/etc/Muttrc`), Mozilla's Firebird (same values as those configured already for Mozilla), etc.

2.3.2. Language tasks

The Debian task selection tool, called **tasksel** is run in every new system installation and allows to define a set of packages aimed for specific purposes.

One widely developed used of tasksel are "language tasks". These are tasks that are installed depending on the installation language. They should feature sets of packages that are specific to the related language.

2.3.2.1. Sets of language tasks

There are currently two language tasks per language: one <language> task which depends on nothing but standard packages, and one <language>-desktop task which depends on the standard desktop task.

Packages in language tasks can include localisation packages for software that are installed by other tasks or by the standard system. For instance, the <language>-desktop tasks install localisation packages for the packages that are part of the standard desktop task (kde-i18n-*, openoffice.org-l10n-*, etc.).

These tasks can also include packages that are needed for the rendering of the given language: TTF or Postscript fonts for <language>-desktop tasks, console fonts for <language> tasks, etc.

2.3.2.2. Maintenance of language tasks

Adding new language tasks should be partly automated by the tasksel maintenance team, i.e. the Debian Installer team. As soon as a new languages appears in the D-I supported languages, an equivalent

language task should be added, if at least one specific package motivates it (for instance a localisation package for on of the packages that are part of the desktop tasks.

Translation teams and translators should also be active by proposing enhancement to tasks related to their language.

New tasks should be reported as wishlist bug reports again the tasksel package.

2.3.3. Translation packages

This section will be completed in further releases of this paper

2.4. Translation of documentation

Users of an operating system require, in order to be able to use it, both online and offline documentation.

In UNIX systems, online documentation is typically handled through manual pages (retrieved through the use of **man**)[3]. In Desktop environments both KDE and GNOME provide online help systems.

Offline documentation includes documents such as the Debian Installation Manual, the Debian Reference Guide, or the Security Debian Manual. These manuals, developed by the Debian Documentation Project (http://www.debian.org/doc/ddp) (DDP), are typically printed by users and, sometimes, read from other (non-Debian) systems at the project's website. Some of the manuals are included in the official CD set and most of them are also available through Debian packages.

Obviously, users not proficient with English will prefer documentation in a language they're familiar with. This is why translation of documentation is needed.

2.4.1. SGML/XML documentation

The *DDP* project develops documentation for Debian written, primarily, in SGML format. The first documentation produced by the project was written in a variant of SGML called *debiandoc-sgml* for which tools were developed to convert the SGML documents into different formats (HTML for online viewing, PDF and PostSCript for printing, and simple text). The latest documentation written by members of the project or revisions of available documentation (such as the Debian Installation Manual) has been written using XML and more specifically Docbook-XML which is widely used by many free software projects[4].

The translation of documentation written in SGML or XML format, however, faces some initial issues:

- SGML documents are usually written in a single (big) file. Since translation teams are typically created in order to handle (small) documents this means that the translation coordinator has to break up the original file in chunks;

- Translators cannot directly use the tools they are used to such as tools intended to work on PO files;

- Automatic publication of documentation in HTML format of documentation needs to be adapted so that they are capable of publishing both the original document and the translations without one conflicting with each other;

- Automatic publication of documentation in PS/PDF needs to be able to handle non-European character sets (i.e. specific fonts) properly;

- Tracking of changes and updates in SGML/XML documents needs to be done through source diffs (which requires access to a revision control system). If a single file is being used, tracking differences will be more difficult;

- Diffing tools handle poorly the fact that text can be wrapped to different column lengths and paragraphs might be "moved" around without actually changing context.

In order to overcome these issues, the document writers, in cooperation with translators, have introduced significant changes to how documentation is written:

- SGML documents are broken up into smaller files (typically one file per chapter);

- New tools have been introduced to be able to convert SGML/XML documents into PO files: **po-debiandoc** (now deprecated), **po4a** (see Section 4.3.2) and **poxml**;

- New tools have been introduced to be able to detect when files within a document have changed: **doc-check** (see Section 4.5) and track the specific changes through the revision control system;

- The publishing toolset (based around `Makefiles`) has been adapted in order to build and publish both the original file and the available translations.

Currently, the Debian Documentation Project uses the CVS server at cvs.debian.org[5]. Documentation is compiled through a set of Makefiles and published on the official project web site (http://www.debian.org), which updates its copy of the CVS repository and compiles all the documentation daily. The CVS server at cvs.debian.org holds most (but not all) of the documentation available. The most notable exception is the *Debian Installer Manual* which is available in the SVN repository of the Debian Installer project on Alioth (svn://svn.d-i.alioth.debian.org/svn/d-i/trunk/manual) (web access to the SVN repository (http://svn.debian.org/wsvn/d-i/trunk/manual/?rev=0&sc=0)). The development version of the Debian Installation Guide is available at the Debian Installer project's web pages (http://d-i.alioth.debian.org/manual/) also on Alioth.

Translators either get access to the CVS or use the original documentation maintainer as a proxy to publish the information in the CVS. When a translation is added to an available document the maintainer typically needs to update the *LANGS* (or *LANGUAGES*) variable in the document's `Makefile` in order to tell the publication system to also build copies for that language. If the added translation builds, it should be available in the Debian website after the next daily build.

In order to track the changes of documentation and when a translation needs to be updated, document maintainers and translators use the **doc-check** tool. See Section 4.5.

2.4.2. Debian manpages translation

There are mainly two types of manpages provided within the Debian operating system: those packaged with upstream software (such as binutils or gcc) and those written for Debian tools provided in Debian-only packages. The translation of manpages of upstream software are typically provided either within the package itself or within `manpages-XX` package (where `XX` is a given language codename) the case of the translation of the Linux manpages (http://www.kernel.org/pub/linux/docs/manpages/).

2.4.2.1. Manpages packages

At the time of this writing the following translation manpages packages are available:

- manpages-de and manpages-de-dev - German manpages
- manpages-es and manpages-es-extra - Spanish man pages
- manpages-fi - Finnish man pages
- manpages-fr - French version of the manual pages
- manpages-hu - Hungarian manpages
- manpages-it - Italian man pages
- manpages-ja and manpages-ja-dev - Japanese version of the manual pages
- manpages-ko - Korean version of the manual pages
- manpages-nl - Dutch manpages
- manpages-pl - Polish man pages
- manpages-pt and manpages-pt-dev - Portuguese Versions of the Manual Pages
- manpages-ru - Russian translations of Linux manpages
- manpages-tr - Turkish version of the manual pages
- manpages-zh - Chinese manual pages

Those manpage packages contain, for the most part, the same manpages available in the `manpages` or `manpages-dev` packages although some packages contain extra manpages.

Users will see the translated manpages through the internationalisation mechanisms of the **man** command which will review, when asked to present a manpage, if a translation is available under `/usr/share/man/XX` (with **XX** being the language code of the user's environment[6]). Consequently, the Debian installation system will install both `manpages` and `manpages-XX` through a default installation if the user selects an specific language task for which manpages are available.

2.4.2.2. Issues with manpage translations

One of the main issues with manpages, however, is that there is no provisions in the **man** to detect when a translation is out of date. As a consequence, users reading translated manual pages might be reading out of date content that does not really apply to the latest version of the program's man page.

The translation project is also lacking a central web page where teams can see (at a glance) which manpages are available for translation, which translations are out of date, translated or untranslated and who is the last translator of the manpage.

2.4.2.3. Translation of Debian manpages

The translation of manpages specific to programs developed within the Debian project (such as the **dpkg** or **apt** tools) is a work that falls within the scope of the Debian translation teams.

The translation of manpages for Debian programs are included within the Debian package itself, which means that translators have to request the Debian maintainer to include the translation in it, once finished. Since Debian programs are typically managed through common revision control repositories available to Debian developers or contributors (either at cvs.debian.org or at alioth.debian.org) active translators of a Debian program will typically have access to those resources and will be able to commit directly into the source control zone were manpages are included. It is worthwhile noting, that it is also common for people active in the program translation to work on the translation of the manpage so that the translation of the program messages and options is consistent with the manual page itself.

In order to coordinate the translation of manpages and make it possible to track when the translation changes, the translation teams introduced the `manpages` in the CVS DDP area. This module includes several scripts in order to track manpages translations: **check_trans.pl** and **compare_files.pl**[7].

This module was introduce since translators did not have access to the CVS repository of the programs for which the translations were going to be made available. Consequently, the original manpages themselves could not be modified to include a translation control header to keep track whenever one was modified. In order to keep track of translation status the CVS module holds `INFO` with meta-data of translated documents including:

Manpage

 document's name in the CVS repository, may be different from the one in the source package. This is used as the document ID.

Encoding

 Document's encoding.

Location

> Location of this document in the source package (this value is only set when the source package does contain this document).

Original

> Original ID in the `english/` directory.

Original-CVS-Revision

> CVS revision number of the original document on which a translation is based.

Translator

> Translator's name, is be used to send automatic notifications when a translated man page is outdated.

Original manpages are included in the `english/` directory of that CVS module by the translation teams and need to be updated manually when the original file is updated. Based on the meta-data information and the CVS revisions available for manpages the scripts can track when a manpage is outdated and notify the translator in charge of it.

Unfortunately, this mechanism, initially developed by the French translation team and used by other teams, is not being maintained. There have been no updates in the English manpages for two years and translations have not been updated there either.

FIXME: Describe use of po4a. The CVS-DDP module is not that much used anymore since most translators are now in the packages themselves...

2.4.3. Coordination of documentation translation

Initially, some of the translation projects (French and Spanish) introduced their own documentation translation management system[8] in order to coordinate the translation of the Debian Documentation Project published manuals. This management system was based on a flat database that included the available documents in the DDTP system and the status of translations. With the use of Perl scripts, this database was converted into HTML files that were published on the website so that the translation team could see which documents were being worked on and who was coordinating the translation.

This system was not integrated with the document database provided by the DDP itself and the translation teams have, for a few years, made use of the translation robots (see Section 3.4) in order too coordinate translation of documents themselves.

Notes

1. Translations that are out-of-date for more than six months are automatically removed, regardless of

the number of revisions they are behind. For more information read
http://lists.debian.org/debian-www/2004/01/msg00323.html

2. This web application was developed by members of the Spanish translation team and available at
http://www.laespiral.org/

3. The GNU project prefers the use of **info** documentation but most upstream developers just provide
manpages.

4. Including distributions such as Red Hat GNU/Linux, or the Linux Documentation Project
(http://www.tldp.org)

5. The web interface can be accessed at http://cvs.debian.org/ddp/manuals.sgml/?root=debian-doc.

6. As defined through the LANG variable

7. There is an additional script, **gen_db.pl**, used to generate the wml files used for the translation
coordination database in the web page area

8. The Spanish management system (the status database has not been updated since January 2003) is
available at http://www.debian.org/international/spanish/ltcp/.

Chapter 3. i18n/l10n infrastructure in Debian

3.1. Po Translation statistics

The Debian web site features statistics web pages under http://www.debian.org/intl/l10n. These pages collect statistical data about po-debconf and programs translations.

The statistics are collected daily by a script run under a developer's account on people.debian.org (Denis Barbier account at the time of this writing). These statistics use material gathered by another script run under a developer's account on people.debian.org (Pierre Machard at the time of this writing).

The statistics pages help translators learn about areas that need work such as updates for existing translations or new packages/material needing translation work.

The pages also help translators to grab POT files and start working on new translations as well as PO files and work to complete them.

Even though this system has proven to be highly useful during last years, it still carries a few weaknesses that prevent calling it an overall i18n/l10n infrastructure:

- it depends on scripts running under individual developers accounts;
- it only gives statistics for the *unstable* branch of the distribution and thus does not allow to get statistics about l10n in testing during release preparation;
- it does not track down the status of the work by translation teams and does no point the reader to the existing translation teams. For that reason, teams have developed their own tracking work method (some parts will be detailed in Section 3.4)
- it is not linked to the bug tracking system and does not allow checking whether an incomplete or missing translation has a pending fix in the BTS (and for how long the fix is pending) or in the package development revision control system.

A way to go would be integrating all these needs in a more general i18n/l10n infrastructure.

3.2. Website translation statistics

Statistics for the translation of the Debian web site are collected in a different set of pages, run by different scripts.

This section will be completed in further releases of this paper

3.3. Debian installer translation statistics

In order to track down issues more closely, the Debian Installer (D-I) i18n coordinators have setup translation statistics pages which gather the status of l10n for all core D-I packages as well as packages defined as part of the D-I "levels" of translation (FIXME: reference to DC5 talk).

These pages point to the individual packages RCS directories and archives. They give translators a more accurate view of work to do and immediately reflect applied changes.

They could be enhanced in a few ways:

- integrate them in the main web site. The status pages are generated by scripts running under Dennis Stampfer account. These scripts are not publicly visible and setting a new location for the pages would require Dennis expertise;

- revamp the pages to make the main page less verbose. There is currently a big amount of information on this page and it should be redesigned as a whole web site...or integrated into a more general i18n infrastructure;

- allow getting statistics about all components in their own RCS trees, in unstable and in testing. This would allow getting a better picture of the real l10n status for releases of D-I.

A way to go could actually be integrating these requirements in a more general i18n/l10n infrastructure.

3.4. Translation robots

One of the things that typically takes more time when coordinating translation projects is keeping track of what is each member of the team working on, and what is the precise status of each of the translations.

The most well known translation robot is the one used by the Translation Project (http://translation.sourceforge.net/). This is an e-mail service that takes care of PO file submissions for translations registered within the project. It checks if files sent to it are receivable, that is, if a translator has filled the translation disclaimer [1]. This robot also calls **msgfmt** to see if the PO file is healthy. The robot also sends notices of updated PO files to the translation teams whenever translations need to be updated.

This translation robot however, only tracks PO files, and only PO files of those GNU projects registered with it. Bearing in mind that the requirements of the translations teams in Debian were different, the translation teams started writing translation robots to handle their own translation process. This work was started by members of the French team and then reused by other translation teams including Spanish, Dutch, Brazilian Portuguese, and German.

The translation coordination robot is an e-mail robot that "listens" to the mails sent to the translation teams mailing lists (debian-l10n-XXXX) and looks for a set of pseudo-urls in the messages' subjects. These pseudo-urls are composed of a translation status and a translation item (see Section 3.4.2). The robot takes this information to compose a list of items being translated, the status of the translation and the translator in charge of it.

This is a useful tool for both translation coordinators and members of the translation teams. Any member can, at a glance, see the status of a translation and help if help is needed (for translations or reviews). It also helps people detect translations that are stalled (a translator stated that they were going to work on it, but didn't finish actually finish it).

Currently, there are several translation robots in place. First, there is a generic robot (http://people.debian.org/~bertol/) that handles different mailing lists by crawling the web site information with a set of scripts (http://people.debian.org/~bertol/scripts/dl10n/). This robot is used by most translation teams, although some teams use their own robot, currently active are the: Spanish team which uses its own robot, available at Spanish translation coordination robot (http://www.debian.org.es/cgi-bin/l10n.cgi?team=es), Dutch translation coordination robot (http://dutch.debian.net/), Catalonian translation coordination robot (http://ca.debian.net/). These last robots, instead of crawling the web site, use real e-mail addresses which are subscribed to the team's mailing lists and handle messages in real time through procmail filters that filter this information to the translation robot's status database.

3.4.1. Translation stages

Translations evolve through the following stages:

- First, a translation needs to be done when there is a new (untranslated) item or document. Typically, the translation team coordinator asks somebody in the team to work on it;

- Then, a member of the translation team answers by saying that (s)he will work on that item;

- Once the translator finishes the translation (s)he sends the item for review to the translation mailing list;

- After several reviews by peers, the translation is finally changed and a new version is sent for a final review;

- The translator then picks the final version and submits it to the upstream maintainer. This is sometimes a package maintainer, but it can also be somebody with access to the CVS where the translations are maintained. Typically, for some translations, this step involves translators using the Debian Bug Tracking System to contact the maintainers (see Section 4.6);

- After some time, upstream maintainers incorporate the provided translation in the package (or CVS) and the translation is considered "published". This is the final step for a translation, until the original translated item changes.

When the original item (document, wml or PO file) is modified, the cycle starts again. However, in that case, the last translator who worked on it is considered upstream's point of contact ((s)he should be

contacted whenever there is a need to update the translation). The last translator is also typically considered as the maintainer for upstream's translations so there is no need to tell the translation team that (s)he will be updating the translation ((s)he is supposed to do it, as (s)he is in charge). Moreover, on many occasions, when the changes to the original item are few, there is not really a need to do a full review of the new translation by the translation team.

3.4.2. Pseudo-urls used by the robot

A pseudo-url is made of the following:

```
[<state>] <type>://<package>/<file>
```

These pseudo-urls are used in the mail Subject: to help the robot distinguish which mails need to be handled by it and which mails are part of the mailing list discussion.

The contents of the pseudo-url are:

state

   The state the translation is in, for more information see Section 3.4.2.1.

type

   The type of item being translated. The translation coordination robot accepts the following item as valid types in the pseudo-url to indicate a translatable item: po-debconf, debian-installer, po, man or wml (webwml is deprecated, wml should by used instead).

package

   the name of the package where the document came from. www.debian.org is used for the wml files of the Debian web site cvs.

file

   the filename of the document, it can contain other information such as the path to the file or the section for a manpage, so no other document in the same package should be referred the same.

The structure of name depends on the chosen type. In principle it's just an identifier, but it's strongly recommended to follow the following rules:

• po-debconf://package-name/language.po

• po://package-name/path-in-sourcepackage/filename.po

• debian-installer://package-name/path-in-sourcepackage

• wml://www.debian.org/address_of_page

• man://package-name/section/subject

3.4.2.1. Translation states handled by the robot

The translation coordination robot can track what stage is a translation for any item through the use of the following keys in the "state" part of the pseudo-url:

TAF

> ("Travail À Faire", French for "translation to do", "taf" also means "work" in French slang) is sent to indicate that there is a document that needs to be worked on;

ITT

> (Intent To Translate) indicates that there is a translator that is planning on working on a given translation. This helps preventing translators to duplicate their work;

RFR

> (Request For Review) states than an initial translation is finished. The translator will attach the translation itself to the sent e-mail, for peer review. This key might be used more than once for the same item[2] if substantial changes have been made to the initial translation based on other's comments. Just like with CVS commit e-mails, translators expect a reply to these requests even if it's just to say "The translation is OK.";

ITR

> (Intent To Review) a peer of the translation team notes that (s)he is working on a review of the translation and might take some (typically because the translation is large, or because the reviewer will not have time available until a given point in time) This is used to prevent the original translator to consider the translation as finished (send an LCFC, see below);

LCFC

> (Last Chance For Comments) tells the team that the translator considers the translation to be finished and has included the comments from the review process. (S)he is giving a last chance for peers to review before the translation is submitted upstream. Typically, it is sent when there are no no ITR's, discussion following the RFR has ended and it has been three days since the RFR was sent. Most translation teams don't allow translators to do this unless at least one member of the team has reviewed the translator's work;

BTS#<bug number>

> (Bug Tracking System) tells the team that a bug has been open to submit the translation to the maintainer. This is useful, since the translation robot can then automatically start checking if the bug report is still open and updates the translation status accordingly;

FIX#<bug number>

> (bug FIXed) notes that an open bug has been fixed already (useful if the translation robot missed it being closed);

DONE

> states that the translation has been finished and is now included upstream. This should be used in cases where no bug report is involved such as web site translations. Otherwise, the robot will handle

DONE automatically by crawling the Bug Tracking System;

HOLD

> put a translation on hold, when the original version has changed but there is no need to update the translation, e.g. the translator knows other modifications will be done soon on the translation and they don't want someone else to update it too quickly.

3.4.3. Example of translation robot usage

This is a typical example of the way the translation robots are currently used.

- A translator (T) wants to work on the PO-debconf translation of the exim4 package, instead of just saying so in natural language in the mailing list. (S)he sends this:

  ```
  Subject: [ITT] po-debconf://exim4
  ```

  The translation robot, when seeing that format in the mail's subject, processes the mail, retrieves the data (it is an ITT, of the po-debconf type, for the exim4 package, send by translator T on this date and time), stores it in a database. This information is presented whenever the translation status page is viewed (since it's driven by the database). The body of the mail is not processed, only the subject.

- Translator T completes the translation a few days later and wants people to review his/her work, so (s)he sends the following mail:

  ```
  Subject: [RFR] po-debconf://exim4
  ```

  The file to review should be attached to the mail[3]. The robot processes this mail, notes the status change in the database and extracts the file from the mail. The web pages can be used to retrieve the file itself.

- After some reviews, translator T sends a mail with an LCFC, attaching the file (which is, again, parsed by the robot) and after a few days sends the document to the BTS and sends this mail to the list:

  ```
  Subject: [BTS#123456] po-debconf://exim4
  ```

  Once this is done, the translator's work is considered finished and the translation robot will, through a periodic job, review if the bug is closed in the bug tracking system. Whenever it is closed, it will be marked as DONE (and will be hidden from the view of the page after a month)

Throughout this process both members of the team subscribed to the list, new members which were not subscribed when the process started and the translation team coordinator have full access to the translation status (and its history) through the web application of the translation team robot.

3.4.4. Future changes for translation robots

In the future, the different translation robots of the translation teams should be merged into one common database as part of Debian's infrastructure for translators. This would help prevent having robots coded in different ways and help that new features (such as handling compressed translations or testing PO files with **msgfmt**) need to be coded in each of the independent robots.

Notes

1. Translators of the Translation Project have to send, through postal e-mail, a form that disclaims in writing by the translators, before being accepted for inclusion in the distribution. For more information, read http://translation.sourceforge.net/HTML/disclaim.html

2. Some translation robots use RFR2 for subsequent reviews

3. Some translation robots don't handle compressed files but most will handle MIME attachments

Chapter 4. i18n/l10n tools in Debian

4.1. Generic tools: gettext

In order to be able to internationalise the user environment the GNU project developed a set of tools which are known as **gettext**. These tools make use of the **locale** definition that is a part of the POSIX.2 standard and is implemented in the GNU **libc**. These definitions include the basic tasks of representing currency formats, date or numbers. But they also include the definitions of sorting tasks and the classification of codes based on the user's culture (such as the order of the letters in the alphabet). This aspect of a user's environment only needs to be defined once since these definitions, once covered, do not typically vary throughout time (except when fixing bugs or introducing new languages).

The translation of messages according to the user's language in any given interface is, however, a more variable aspect of internationalisation of software. Not only messages that are shown to the user are translated usually, error messages, help of the options that the program use and any other message that a user might see at any point in time is suitable of translation.

The **Gettext** tool was developed within the GNU project between 1994 and 1995 by a diverse group of programmers. This tool helps the creation of programs that can be distributed with multiple message catalogs in different languages. In localised environments the programs will choose the message catalog most suited to the environment as defined by the user and present messages in a given language.

This tool is almost transparent to the programmer. The programmer just has to mark in a special way messages that need to be translated. It is also transparent to the translator since the precise location of messages in the source code and the relocation of messages. Translators just have to keep the translation of a list of messages current. The gettext tools handle the creation of catalogues and their regeneration when the sources change but, at the same time, preserving translations already done.

This way, the work of translating messages within a program is reduced to the task of doing an initial translation of all the messages and then the maintenance of the small (or big) changes introduced in the code that might have as a consequence the apparition (or change) of messages. The programmer has just to prepare the sources to use gettext through the use of the tools both in the program itself and in the tools that compile and build it (a process known as software internationalisation). Once this is done, the work of both groups can be done independently, which helps development both ways. That is, a translator does not have to depend on the programmer to introduce a new language and a programmer does not have to wait for the translators to do their job in order to publish a new release.

More information is available in the official gettext project pages (http://www.gnu.org/software/gettext/gettext.html). If gettext is already installed, online documentation is available that can be read executing **info gettext**.

Translation of messages using gettext is very simple. A translator just needs to pick up a PO file either

partially translated or completely untranslated and fill in the missing "gaps". Having many people with many different capabilities willing to cooperate makes it possible to collaborate without the need of in depth knowledge of the translation mechanism. This is how translation teams born. Several front-end interfaces to gettext are available, such as **gtranslator**, **kbabel**, **poedit** or emacs'po-mode. One of consequences of the availability of the **gettext** tools is that it is not necessary to have programming knowledge to work on translation. The only requirements for working in translations are the knowledge of the original language, the language it will be translated to and the ability to edit .po files (with multiples tools being available to ease the task as described above).

- PO file example

```
# Copyright (C) 1999 Free Software Foundation, Inc.
# Javier FernÃ¡ndez-Sanguino PeÃ±a<jfs@computer.org>, 1999.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 1999-06-21 14:21+0200\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: Javier FernÃ¡ndez-Sanguino PeÃ±a <jfs@computer.org>\n"
"Language-Team: ES <ES@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: ENCODING\n"

#: hello.c:86
#, c-format
msgid "Usage: %s [-htvm] [--help] [--traditional] [--version] [--mail]\n"
msgstr "Modo de uso: %s [-htvm] [--help] [--traditional] [--version] [--mail]\n"
#: hello.c:148
msgid "This is GNU Hello, THE greeting printing program.\n"
msgstr "Este es Hola de GNU, EL programa que imprime un saludo.\n"

#
msgid "Translate this"
msgstr ""
```

The GNU project supports different groups of internationalisation. These groups are coordinated by a person in charge of the translation team. There is no requirements, however, to provide constant dedication within a translation group. Translation can be a discrete effort. The existence of these groups, however, guarantees the revision of these discrete efforts from anonymous contributors (many time users). These groups are also in charge of developing glossaries to make translation of programs uniform and to maintain and update translations whenever new programs are published.

This work is coordinated through the *Free Translation Project* at http://translation.sourceforge.net/. The

state of translations can be reviewed through a database of translations and translators at
http://translation.sourceforge.net/cgi-bin/registry.cgi?team=index.

4.2. Translation headers: wml

This section will be completed in further releases of this paper

4.3. Po for everything (po4a)

po4a (http://po4a.alioth.debian.org/) (*PO for anything*) is a set of tools originally developed by Martin
Quinson with the goal of easing translations and, more interestingly, maintenance of translations,
through the use of gettext tools in areas where they were not expected, like documentation.

Tools in the po4a package are used to update PO files from the original files and generates translated files
from these PO files.

The usual process is made of one initial step (convert translations to PO files) and two recurrent steps
(update PO files and regenerate translated files).

4.3.1. Converting existing/translated documentation to PO

Converting documentation files to PO is the job of the `po4a-gettextize` command. When starting a
new translation, `po4a-gettextize` will extract the translatable strings from the documentation file and
write a POT file from it. When translated files exist, strings are grabbed from them and collected in a new
PO file.

The `po4a-gettextize` tool only extracts the Nth string from the translated file and matches it to the
Nth string of the original file in the created PO file. It verifies that the original document and the
translated documents have the same structure, but can't detect the state of the translation.

Because some manual expertise by translators is then required, all extracted strings are marked as
"fuzzy" by the `po4a-gettextize` process.

4.3.1.1. Converting manpages to PO

Converting existent manpages to PO files can be done using the **po4a-gettextize** tool:

```
$ po4a-gettextize -f man -m the_manpage -p foo.po
```

Or if an existing translation exists:

```
$ po4a-gettextize -f man -m the_manpage -p foo.po -l existing_translation
```

PO files can be converted back to manpages with:

```
$ po4a-translate -f man -m the_original_manpage -p the_PO_file -l translated_manpage
```

Upstream authors can use the following Makefile to translate manpages by placing it in a `po` subdirectory under the documentation directory (which holds the manpages in troff format).

- po4a Makefile example

```
PROJECT=project_name

# Sections of the manpages
SECS=1 2 3 4 5 6 7 8

PO4ATRANSLATE  = po4a-translate
PO4AUPDATEPO   = po4a-updatepo
PO4AGETTEXTIZE = po4a-gettextize

MAN1 = $(notdir $(basename $(wildcard ../*.1)))
MAN2 = $(notdir $(basename $(wildcard ../*.2)))
MAN3 = $(notdir $(basename $(wildcard ../*.3)))
MAN4 = $(notdir $(basename $(wildcard ../*.4)))
MAN5 = $(notdir $(basename $(wildcard ../*.5)))
MAN6 = $(notdir $(basename $(wildcard ../*.6)))
MAN7 = $(notdir $(basename $(wildcard ../*.7)))
MAN8 = $(notdir $(basename $(wildcard ../*.8)))

# List the languages the manpages are translated to
LANGS = fr
TRANS=$(foreach lang,$(LANGS),\
          $(foreach sec,$(SECS),\
                  $(foreach man,$(MAN$(sec)),$(man).$(lang).$(sec))))

PARTIALPOT=$(foreach sec,$(SECS), $(foreach man,$(MAN$(sec)),$(man).$(sec).partial.pot))

all: $(TRANS)

#### CREATION OF THE POT FILE ####

$(PROJECT).doc.pot: $(PARTIALPOT)
    msgcat --use-first $^ -o $@

%.partial.pot: ../%
    $(PO4AGETTEXTIZE) -f man -m $< -p $@

#### CREATION OF THE POT FILE ####

$(PROJECT).doc.pot: $(PARTIALPOT)
    msgcat --use-first $^ -o $@

%.partial.pot: ../%
    $(PO4AGETTEXTIZE) -f man -m $< -p $@

#### TAKE CARE OF THE PO FILES ####

%.doc.po: $(PROJECT).doc.pot
    @echo -n "Merging $(PROJECT).doc.pot and $@: "
    @msgmerge $@ $(PROJECT).doc.pot -o $@.new
# Typically all that changes was a date. I'd prefer not to cvs commit such
# changes, so detect and ignore them.
    @if diff -q -I'#:' -I'POT-Creation-Date:' -I'PO-Revision-Date:' $@ $@.new >/dev/null; then \
```

4.3.2. Maintaining translated documentation with po4a

The dataflow can be summarised as "master document --> PO files --> translations". Any changes to the master document will be reflected in the PO files, and all changes to the PO files (either manual or caused by the changes from the master document) will be reflected in translated documents.

Po4a tools allow defining a minimum translation ratio below which the translated document will not be generated anymore or will only contain the original strings. This avoids generating documents with too few translated strings but still allows the publication of complete translations. As a consequence, translated documents may still contain some strings in the original language. This tries to guarantee the accuracy of the translated document.

This behaviour is a big improvement for documents such as manual pages. Indeed, an argument often used for not using these translations is the lack of guarantee that the translations are in sync with the original man pages. The po4a tools lower this argument, the price being sometimes mixed man pages, though.

4.3.3. Recommended po4a organisation for software

This section is mostly taken from the **po4a** documentation and describes the recommended organisation for software that use **po4a**.

A standardised architecture of the source tree will help the translation teams when they try to detect the POTs that need to be updated. As a consequence, the following architecture is recommended:

```
/
/doc/
/doc/en/
/doc/en/
/doc/po4a/
/doc/po4a/add_<ll>/
/doc/po4a/po4a.cfg
/doc/po4a/po/
/doc/po4a/po/<pkg>.pot
/doc/po4a/po/<ll>.po
/doc/<ll>/
```

Or, if you want to avoid a big POT and split it according to the packages, documents, formats, or subjects, you can use the following architecture:

```
/
/doc/
/doc/en/
/doc/en/
/doc/po4a/
```

```
/doc/po4a/add_<ll>/
/doc/po4a/<pkg1>/po4a.cfg
/doc/po4a/<pkg1>/po/
/doc/po4a/<pkg1>/po/<pkg1>.pot
/doc/po4a/<pkg1>/po/<ll>.po
/doc/<ll>/
```

It is important to avoid a build failure if a translation cannot be generated (the PO is too outdated, an addendum cannot be applied, ...). Wildcards should therefore be used or tests be included to check whether the files are generated in the 'install' or 'dist' rules.

When po4a is used upstream, it is recommended to run po4a in the 'dist' rule. This will update the POT and POs, and will generate the translated documents.

These translated documents can be distributed in the source archive if the maintainer doesn't want to add a build dependency on po4a. This requires adding an autoconf check on po4a and will allow updating the documentation if po4a is available locally. If po4a is not available, documents will be distributed without being synced with the original version, but the build process won't fail.

It is important to distribute the POT and POs in the source archive.

When po4a is used in a distribution such as Debian, one should ensure that the source of the package contains only up-to-date POT and POs. This means that po4a should be run in the 'clean' rule of the make process (typically in **debian/rules** for Debian packages.

```
clean:
        # Update the POT and POs
        cd <...>/po4a && po4a --no-translations --rm-backups <package>.cfg

build:
        # Generate the translations
        cd <...>/po4a && po4a --rm-backups <package>.cfg
```

4.4. Debconf-updatepo and po-debconf tools

Tools in the po-debconf package are mostly aimed for maintainers use. The central tool is the debconf-updatepo utility. This utility should be run each time a change happens in debconf templates, i.e. most often the files names *.templates in the debian/ directory of the package source tree.

debconf-updatepo looks for all files listed in the `debian/po/POTFILES.in` and search in these files for translatable strings. Instructions on how to set a debconf string as translatable are given in the

po-debconf(7) man page. This can be summarised as "just put an underscore character before the field names".

All translatable strings are written by **debconf-updatepo** in a "template file" named `debian/po/templates.pot`. In the same time, all PO files that are present in `debian/po` are updated with regards of the new strings.

Changed strings are marked *fuzzy* (thus keeping the old translation) or *untranslated*. In such case, the old translation is kept as *obsolete* entries et the end of the PO files.

"fuzzy" strings translations are never used. When a debconf template includes more than one fuzzy string, the whole template will be shown untranslated to users.

Maintainers should run **debconf-updatepo** as soon as they change templates. Some i18n maintainers recommend running it in the package clean target to ensure that all PO files AND the POT file are up-to-date. There is nothing worse than a package with obsolete files. Some other maintainers rule against running **debconf-updatepo** in the clean target as this is likely to modify files in the debian/ directory.

Rebuilding the templates file that will be shipped with the package, as of `debian/<package>/DEBIAN/templates` is the job of **po2debconf**. This utility is auto-magically called by **dh_installdebconf** for developers that use debhelper tools. This is actually a very good argument for using debhelper tools for packages that use debconf.

The final templates file uses the encoding defined in `debian/po/output`. The default value is "utf8". It is highly recommended to use "utf8" in all cases. There is actually no good reason to use anything else. This will not affect the way the templates will be displayed, even in non UTF-8 environments.

4.4.1. The podebconf-report-po utility

The **podebconf-report-po** utility is aimed to send notices about needed updates to all translators who have incomplete files (at least one fuzzy or untranslated string).

Using it before planning an upload of a package with modified templates is strongly recommended. Translators do not really like to discover changes and needed updates after a package has been uploaded.

When using this utility, maintainers should remember that many translation teams need time for their internal QA processes to take place. Leaving only a few days for translators to update their work, especially when important changes occurred, is nearly similar to a call for bad translations.

4.5. Documentation's doc-check

Translation updates for documentation are handled through a script called **doc-check**. Although the tool shares a common name, different documents use a different version (or incarnation) of the same tool (or concept). Basically, the translators need to add to each translated file a translation header that specifies which file revision was translated. So, if a document was translated based on the 1.12 revision in CVS, a translator would add this header:

```
<!-- CVS revision of original english document "1.12" -->
```

The specific header format used might vary between documents, for example, this is the translation header for the translation of a file in the Debian Installation Guide:

```
<!-- original version: 12756 -->
```

If these headers are in place a translator can use the **doc-check** tool to:

- show which translated files have a different revision number from the original English files
- show what changes have been made in the original English file
- help keep track of which files have not yet been translated

In order to use the tool the translator just needs to update his/her local copy of the document source code, run the script and review the output. The use of a revision control system helps the translator extract[1] which changes have been introduced in the translated file. (S)he can then update the file, update the translation header and commit his/her changes.

This mechanism is indeed very similar (if not the same) to the mechanism used in the website to manage translations.

Translations that feel more comfortable using PO files can use either po4a (see Section 4.3.2), or **poxml**. The Debian Installer uses **poxml** and provides tools to convert from the XML files to PO files for the translation teams interested. A different set of tools converts the PO files back into XML files. "Out of dateness" of translations is handled through the same PO mechanisms (*fuzzy*) as those used by **gettext** to determine whether to show a translated message or not.

4.6. Use of the BTS for translation work

4.6.1. Translators

As always, using Debian Bug Tracking System to deal with translation work and particularly to send translation updates or new translations, is the recommended way.

Translators are encourage to send new translations or translation updates as bug reports against the relevant package. The use of the **reportbug** utility is encouraged for translators working on Debian systems or Debian derivative systems (such as Debian-Edu or Ubuntu). The **reportbug** is part of the **reportbug** package which can be installed with the following command line:

```
aptitude install reportbug
```

This should be done as root or, if the current user is listed in *sudoers* by using the **sudo** command (this is useful on Ubuntu systems where the root user is not activated by default):

```
sudo aptitude install reportbug
```

Bug should be reported against the *source* package which the translation belongs to, preferably.

reportbug will prompt for a bug *title*. By general agreement, semi-formalised bug titles should be sued:

```
[INTL:<code>]: <language> <type> translation
```

where *<code>* is the language ISO-639 code, *<language>* is the language name in English and *<type>* is the type of translation (which can be "po-debdonf", "programs", "man pages", etc.)

reportbug will request for the bug severity to use. By general agreement in the project, the *wishlist* severity should be used.

reportbug will also prompt for *bug tags* which is a common way to categorise bug reports. The *l10n* and *patch* bug tags should be used.

reportbug then drops the user into a text editor to fill in the bug text. There is usually no need to be very verbose there as most informations are obvious.

Finally, the translation file should be attached to the bug report (**reportbug** is clever enough to warn users when a bug tagged *patch* is sent without any file attached. The attached translation file can be sent as is but several translators commonly compress these files with **gzip** to avoid maintainers mail user agent messing up with the file encoding.

A recent announcement (http://lists.debian.org/debian-devel-announce/2005/09/msg00002.html) by Denis Barbier (FIXME: URL) proposed to use bugs usertagging to categorise l10n bugs.

This proposal sets up user tags such as *l-<code;>* where <code;> is the language code, *l-* being a prefix for *language* while *t-* could be a prefix for *territory* (or country). The *user* to use is *debian-i18n@lists.debian.org*. This proposal is currently not completely finalised and could change in the future.

4.6.2. Package maintainers

Maintainers should use the sent files as is. They should NOT change them and, for instance NOT re-encode them. It is recommended that maintainers check that the translation is sent by someone who is either the former translator or someone with his/her approval. When in doubt, advice should be requested in the debian-i18n mailing list.

When maintainers receive new translations or translation updates for an upstream program, they must send them to upstream and possibly setup a good working method with their upstream for this translation updates flow. The translations of software in Debian should idealistically not be different from upstream translations.

Translation updates are registered exceptions to freezes that occur at the end of the Debian release cycle, so maintainers should make use of that general freeze time to make sure they have complete translations and possibly request for updates to their translators.

Notes

1. The **doc-check** script of the *d-i* project can extract the changes between revisions by making the proper queries to the revision control system if asked to.

Chapter 5. Relationship with other projects

This section will be completed in further releases of this paper

5.1. Translation packaging

This section will be completed in further releases of this paper

5.2. Handling of bug reports for upstream translations

The situation varies considerably between upstream projects which have a strong and well organised upstream i18n/l10n team, such as OpenOffice.org, KDE, Gnome and similar projects, or even "smaller" projects where translations are handled through the Translation Project, and other projects for which the localisation is mostly done by direct interaction between the upstream maintainer and a variable set of volunteers.

For the first kind of project, it is recommended that Debian maintainers and translators avoid interfering with the upstream i18n/l10n resources. This means that translation updates coming through the Debian BTS should be redirected to the upstream i18n team, or to the Translation Project. In short, sending translation updates through the Debian BTS is here discouraged.

For projects where internationalisation and localisation is done autonomously, Debian maintainers can interact more closely with upstream. Most upstream maintainers may be deeply interested by the big amount of Debian resources for localisation and it is recommended to push for Debian translators to maintain as many upstream translations as possible, for such projects.

This needs a closer interaction with the upstream maintainers, for instance by requesting access to his/her revision control system or have him/her deal directly with the Debian BTS (tagging and triaging bugs). Such method has already proven to be a good method to develop a general stronger interaction with upstream, not only for localisation.

5.3. Handling of errors in upstream translations

This section will be completed in further releases of this paper