Preparing a Free and Open Source Software Localisation and Deployment Project

This document attempts to be a step-by-step guide for project managers on how to create and write a FOSS localisation and deployment project for a country or culture. It is specially indicated for countries in which no localisation of proprietary software has yet taken place, but many of the ideas should also be of interest for other developing countries which have an interest of reducing their dependency on foreign proprietary software vendors, as well as the associated costs and security concerns.

Version 0.6 – 24/12/2005

Introduction

In order to enter a digital world without forfeiting its culture, a country must do it by using software in its own language. Software in a foreign language exacerbates the digital divide, makes basic computer training difficult and expensive, impoverishes local culture, and blocks computer-based government processes, as the local language script cannot be used in interfaces and databases.

Proprietary software companies (such as Microsoft) are not interested on translating and adapting their software (what is commonly know as localising it) to economies that are not considered as profitable markets (now or in the future). The only opportunity that the poor economies have access to - in order to have software in their own language - is the localisation of Free and Open Source Software, something that they can do by themselves without having to ask permission from anybody, nor needing to spend astronomical amounts of money.

Localisation of Free and Open Source Software (FOSS) is a phenomenon that has mostly happened inside the world of technology, pushed by technical people or organisations who were already involved in the development process for this software. Also, localisation of Free and Open Source software - because of the technical needs involved up to now - has first taken place in developed countries in which basic proprietary software (such as word processing, spreadsheets, etc.) is already available in local language.

Now the localisation processes has become simpler, and information about how to localize grows, permitting smaller countries which do not have any software in their local language - nor highly trained FOSS technicians - consider FOSS localisation seriously. For these countries, the social advantages of using software in local languages are very large and easy to measure:

- Allowing the government to do its work - using computer technology - in its own language.
- Permitting computerisation of SMEs.
- Drastically reducing training time for people who need to use computers, reducing therefore training costs.
- Allowing younger people to start learning how to use computers (in schools).
- Allowing people out the capital cities to have similar access to computer training that more favored citizens, even if they have not had the opportunity to learn English.
- Recovering the local language as the work language, in a modern office environment that uses computers, avoiding an excessive entrance of foreign terms from the use of computers in a foreign language.

The localisation of FOSS opens the way to development in the same way that building roads does. Local language software must be considered as part of the ICT infrastructure that a country needs. As with roads, the use of infrastructures that do not have a price-tag entry barrier end up producing sustainable development.

Organisations approaching localisation in small economies must look at it from a global perspective, considering all aspects that will lead to the use of localised software. In many cases, they must look at localised software as a “product” that must reach the market, and which will follow the same rules and procedures that any commercial product, and will be subjected to market dynamics.

The work for the localisation of FOSS software must therefore be planned from the beginning, in the form of a project that must consider its goals, benefits, beneficiaries, needed resources, budget, time-frame, packaging, distribution channels, training, etc.

This document attempts to be a guide on how to develop such a project, covering all the above mentioned aspects. The project produced by using this guide can then also be used as a document to search for the necessary funds for accomplishing it.

The guidelines and examples of this manual should be taken as such. The project should be developed based on the reality of the country or region that it addresses. The writer should always remember that his document will be read by people who do not necessarily have an understanding of technology, but who can recognize the social problems that can be solved, and consider localisation as the correct solution, specially if there is a clear road-map to the solution.

The project has to build the case in the head of the reader. This reader first needs to recognize the problem (description of the present situation), understand what you propose as a solution, and then clearly see the social benefits and new stages of development that the solution will bring. The technical details are also important, but secondary to the social issues. They show that the project can be done, while indicating possible time-frame, cost and risks.

The main body of this document is a possible structure for the FOSS localisation project document. The order is not arbitrary, but it is “our” order. If you feel that a different order would be better to express your ideas, please feel free to change it... but remember, if you start with a technical part, your target reader may not continue reading, as he will not understand anything. In order to write a good document all ideas have to be yours, either because you write them or because you copy sections (feel free to do it) that you assume as your own. In short, you must have full ownership of the project that you write, and be prepared to defend it.

Each section of the project is explained, with examples in the cases that they might be necessary. The last two chapters offer a very quick how-to guide on how to put together the necessary information for the project, and a list of further references and examples that might be useful in your work.
Document structure

This section includes the index that we recommend for the project document. Please take it as a guideline, not as some mandatory structure that you must follow. This index is later developed in the next section point by point, either explaining what they should contain or giving you suggested text that you might use if it fits with the situation in your country.

Proposed Index

1. Present situation in the country (or region)
   1.1. Social and economic
   1.2. Linguistic
   1.3. Information technology
      1.3.1 Language of available software
   1.4. Social problems produced by lack of local language software
   1.5. Economic problems created by the high price of licensed software

2. Objectives of the project
   2.1. Social and strategic goals
   2.2. Technical goals
   2.3. Strategy to reach the objectives

3. Project Output

4. Benefits and beneficiaries of the project

5. State of the technology to be used
   5.1. OpenSource Software
   5.2. Encoding for the target language
   5.3. Script and Language support in different platforms
   5.4. Locale and collation data.
   5.5. Fonts
   5.6. Translation (glossary)

6. Localisation work that needs to be done (short descriptions)

7. Government support
   7.1. Policy
   7.2. Access to Government migration
   7.3. Access to Government's training centres.

8. Distribution and Deployment
   8.1. Support
   8.2. Vendors
   8.3. Training for end-user trainers
   8.4. Deployment on Windows platform
      8.4.1. Installation Campaign
      8.4.2. Training Campaign
   8.5. Deployment on Linux Platform
8.5.1. Installation Campaign
8.5.2. Training Campaign

9. Development of Free and Open Source Expertise
   9.1. Universities
   9.2. Vendors
   9.3. Private companies
   9.4. Governments
   9.5. Maintenance of localised Software

10. Project plan
   10.1. Organisational structure for the project
   10.2. Calendar
   10.3. Required Human resources
   10.4. Milestones and evaluation
   10.5. Risk factors
   10.6. Future actions once the project is finished.

11. Budget

12. Technical Annexes
   12.1. Language adaptation – Glossary
   12.2. Language support
      12.2.1. International standards
      12.2.2. Fonts
      12.2.3. Script support in software
   12.3. Keyboard
   12.4. Cultural Data (locale)
   12.5. Localisation of Applications
      12.5.1. Software adaptation
      12.5.2. Translation
      12.5.3. Documentation
      12.5.4. Development of training materials
   12.6. Localisation of a Linux User Interface
      12.6.1. Translation
      12.6.2. Documentation
      12.6.3. Development of training materials
   12.7. Packaging
      12.7.1. Linux-based [and Windows-based] distributions
      12.7.2. Installation documentation

13. Informational Annexes (About the organisation, CVs, etc)
Document Development

In this section we will go through each one of the items mentioned in the Index, trying to hint or propose what we consider would be the adequate content. Please take anything that we write just as our opinion, because it might, in many cases, not fit with the reality or needs of your own case.

1. Present situation in the country (or region)

   Any social project must be placed in the context that creates the problems that it attempts to solve. In the case of localisation problems, these problems are socio-economic and linguistic.

1.1. Social and economic

   In no more than one page, this section should describe the state of the country or region, including data on population, Gross National product, per capita rent, etc. It should also specify socio-economic circumstances that might be of relevance to this project. You might want to include data on education, on the job market, on historical events that have strongly impacted the economic situation of country in the last years (war, natural disasters, etc.).

1.2. Linguistic

   Specify languages spoken in your country, including demographic data if possible (percentage of people who speak each language as mother tongue or as secondary language). Explain what scripts they use. It is especially important to define what language is spoken in the workplace and what language is used for teaching in the school system. You should also mention what language is taught as a second language and if it has any penetration in society or in the workplace.

   In some countries, language is a sensitive issue. If this is the case in your country, and is not politically incorrect to talk about it, then you should mention the present political issues related to the language.

1.3. Information technology

   It would be interesting to have data – for this section – on the number of computers installed in the country or region, the number of computers being sold each year, the operating systems (even versions of the operating systems) and other data that might be of interest do define the goals of the project.

   1.3.1. Language of available software

   Specify the language(s) of the operating systems and applications installed in most computers in country.

1.4. Social problems produced by lack of local language software

   This is an important part of the project. These social problems are the reason why this project needs to be undertaken. These reasons might include (but not be restricted to):

   • Software in a foreign language exacerbates the digital divide. Only those who have access to advanced English training can do computer training,
and have access to jobs or further training that require the use of computers.

- Very few people are so highly trained in English that the learning in this language becomes natural. The time to train somebody in the use of computers in a foreign language is usually very long. Students have to learn a lot of English words (words that are not normally used in other contexts, unless you speak English at work everyday), and memorize them, as they cannot relate them to something they know. Besides the long training, this knowledge disappears quickly if it is not used, as it is based in memory and not in linguistic/semantic logic. Therefore, training is long and inefficient.

- Long training is expensive, not only because it requires paying to an institution for many months, but also because it takes a long time to reach a skill that can be used for work, and all this time, no income is generated. Training for this skill does not fit well with the economic reality of the country, becoming a barrier to the less advantaged.

- Computer training cannot be started until a late age. It is not possible in secondary or high school, because of the language barrier (especially in rural areas). It can only be considered seriously after finishing high school. This makes the learning process longer.

- Computer use by government officials is complicated, as the average level of English is very low, the salaries are also very low and there is no special motivation to start a long process of learning English in order to use computers. This makes training of officials long and expensive for the administration, as well as not very efficient, slowing down possible administrative reforms or decentralisation work that is based on information technology. The Government is forced to work in a mix of its own language and a foreign language or script.

- Small and medium size enterprises only start using computers when it becomes very easy and inexpensive. The use of local language reduces the barriers enough for produce computerisation of SMEs.

- Computerisation of SMEs permits correct and transparent taxation, increasing the Governments income, at the same times that makes if fairer for the SMEs, as it is based on more realistic data.

- Attempting to offer the same opportunities – through the use of computers – in rural areas is almost impossible without computers in the local language.

- Impoverishes local culture. The computers being in English (or other language different from the language normally used at home or at work, all the vocabulary used around the computer (keyboard, mouse, spreadsheet, table, etc.) is taken from the English language and English programs. Most of these words could be easily translated to the local language, but it is naturally easier to use what is there already (only through local language software and training can local terminology be spread). This brings a large body of foreign words into the work place, replacing local words, and therefore impoverishing the local language.
and reducing comprehension in new members of the workplace, until they catch up with the computer-originated slang.

- If the script is not Latin script, and it is not implemented correctly in the computer systems; this deficiency blocks computer-based government processes, as the local language script cannot be used in databases. This leads to having administrative processes use transliteration of local characters into Latin characters, which rarely is exact, and generates large amounts of errors. It also implies using the English alphabetical word sorting order, instead of the alphabetical order in the local language, complicating searches and eliminating this other part of the local culture.

Other problems might be related to:

- The political situation, such as independence under an occupation that imposed a language.

- Special national need to standardize the language. Computer spell-checkers can help students learn the correct spelling of words. This is specially interesting in countries where the level of language standardisation is low, with many dialects, or countries where spelling has changed in the last decades or is changing at the moment.

- Minority languages need special treatment for their survival and their use in computers is seen as one of them.

These are just general issues, but there might be many other reason that are particular to a specific country. It is definitely worth it to spend some time thinking about other problems generated by the lack of local language software.

Another whole set of problems relates to the economic situation of the country.

1.5. Economic problems created by the high price of licensed software

The minimum set of software needed by a computer to be operational consists of an operating system and an office software suite. For computers using Microsoft tools, the cost of the Windows XP operating system and the MS Office suite (MS Word, MS Excel, MS Powerpoint and MS Access) comes to around US$560 per computer. This amount may be reasonable (I would not want to get into this discussion) in countries like the United States, where it is a small part of the GPD per capita, but in other countries it might represent over two years of the GPD per capita, making it an unreachable sum for most people.

Not only final-users need to be taken into account. Also the Government needs to buy computers and their software.

The impossibility to pay these prices, for particulars as well as for the country itself, leads to have people and corporations copying the software and ignoring licensing conditions.

For a small country with 20,000 new computers every year, the cost of software, paid to a single company, would be over 11 million dollars a year.

This cost is added to the cost of having to train people with software in a foreign language, which can multiply by five the cost and the time need for basic training.
2. Objectives of the project

2.1. Social and strategic goals

The objectives of the project must be in direct relation with the social and economic problems stated above. They might include, among others:

- Reduction of the Digital Divide, permitting empowerment of the least advantaged.
- Reduction of training needs for jobs that require computer skills by effectively separating the skill of using the English Language from the skill of using a computer.
- Facilitate the computerisation of the Government.
- Protection of the local language from an excessive entry of foreign terms that complicate training and integration in the work place.
- Permit training on technology from early stages in the education system.
- Permit reaching out to rural areas, being able to offer there training similar to that which might be offered in large cities.
- Allow the Government to use user interfaces and maintain data in its own language and script.
- Permit computerisation of SMEs.
- Support linguistic goals of the Government.

2.2. Technical goals

The technical objectives are much more specific, but are not yet the output of the project itself, nor the strategy. In our case they should be something similar to:

- Define your product: To develop or translate computer systems and applications in local language that can be easily used by citizens of the country without having to learn English or any other foreign language. These computer systems should have the lowest possible cost, being free if possible.
- Define the target public for whom these systems are developed. For example: the basic computer user (over 80% of the users, probably) who only office and Internet software, uses the computer to listen to music and might eventually burn some CDs or DVDs
- Define what you want to give to these users: a system that has, in local language, all the programs that they ever use.
- Documentation. Without documentation programs are difficult to use. Include in the goals the development of documentation for these products.
- Define training objectives. These objectives can be defined in terms of amounts of people trained each year to use this system, or in terms of percentage or absolute number of computer-training centers that teach the use of these programs. It can also include the creation of Open Source expertise that will help make the system sustainable. Define the training materials that will have to be developed before training might...
take place. Standardised training materials are important for the transmission of the local language computer terms defined in the localisation process.

- Define distribution goals. This can be done talking about percentage of new systems sold in year xxxx that will have the applications pre-installed or total number computers in which the tools will be installed by such date or in such year. These goals have to be well though and must be possible to attain with the strategy and plan developed below. As an example, this could be:

  The final objectives of the project are to have localised applications pre-installed with 80% all new systems on Windows platform. Also, to have the full Linux-based localised system pre-installed in 20% of new computers sold in the country on the first year after the release of the system, reaching 50% on the second year. In the third year we expect almost 80% of personal computers to include it, even if another operating system (such as Windows) is also pre-installed in these equipments.

- Define other technical goals, such as developing systems and documentation to help the administration to computerize itself in its own language and script.

2.3. Strategy to reach the objectives

The strategy is the high level plan to reach your objectives. It will lead you to attain the social objectives by means of attaining the technical ones. It does not come down to the details of “the plan”. It sets the lines in which the plan has to be carried.

The strategy defines your path from the present situation to a situation in which all your social goals have been reached. A complete strategy must take into account several issues:

**Policy strategy**

Unless you are in the Government, you have little control over ICT policy. You might nevertheless consider lobbying for the inclusion of FOSS and Open Standards in the Government policy. You must decide if you want or not to make this part of your project.

**Platform Strategy**

An important line of the strategy is to decide what platforms will be used in the localisation. It is a pity to have to come to technical explanations in such an early part of the document, but it is difficult to avoid. You will have to decide your platform strategy. Two possible options are:

- The first option is to develop only free software applications running in a free platform (Linux). The advantage of this approach is that all the added value of the local language software is in one platform; therefore the interest of changing to the new platform (Linux) is very high. The negative side is that it requires a 100% change in the computer system, and very few people are prepared for this change up-front, including vendors, who must be an important part of the distribution strategy, as well as employers, who might see danger in decrease of productivity related to changes in computer system.
• The second option is to translate applications and release them in both platforms (Windows and Linux, there are very few Macintoshes in the third world, and their users are normally in specific professions, and not part of our target). This allows users to start using the local language applications without changing the operating system. The cost of change is much smaller, and users are much more open to the new applications, which simplify their work and do not pose the threat of having to do a larger change all together. Once there is a large park of installed local language programs, the change to a Linux platform (for the user) only requires a change in the User Interface, learning how to use a few tools, a small effort that can be learned in a day or two (knowing that the applications are the same they were using until now, with exactly the same interfaces). This type of approach helps enlist the support of computer vendor and the private training industry, as they do not need to make drastic changes in order to help distribute local language FOSS programs. It involves a process of creating Linux expertise in vendors and in the training industry.

Training strategy

Training is a fundamental component of reaching the people, and therefore the objectives. The training strategy attempts to use training as a conductor for your local language terminology and your products, and the trainers as the marketing channel for the new software.

A good training strategy may include:

• Unifying terminology in all training materials used to teach the use of the localised applications.

• Trying to turn existing computer-training centers (private as well as Governmental ones) into distributors and proponents of your local language technology.

Even if this hits already the bottom line of what is strategy and is more part of a plan to fulfill the strategy, a possible course of action on training can be the development of a certification program, in which an entity (your organisation) develops standardised training and evaluation materials, and issues a certificate to students who follow such training and pass the evaluation. Certification courses can only be imparted by certified instructors who have been trained by your organisation (or a partner). Computer training centers that have certified instructors and offer the courses can also be certified centers. All this leads to controlling the training materials, and therefore the training terminology, at the same time that training-centers themselves are part of the marketing strategy when they announce the courses. It is even better to have the Government be involved in the certification system, issuing official certificates. The training materials must also be Open Source, otherwise their purchase becomes a barrier for trainers and their institutions.

Distribution strategy

The most important distributors of software are usually computer vendor, who tend to sell computers to their customers with all the necessary software already installed. Software vendors also need to be taken into account, and public Internet access places are also very good points for allowing people to see, test
and use the programs that have been localised. Don’t include any plans to contact them here, just mention that reaching them is part of your strategy, and add any other strategic ideas that you might want to include.

A good distribution strategy needs to address all these three sectors, adding to them NGOs who are working in the world of technology and provide training or use of computers in provincial or rural areas.

**Marketing**

Marketing itself is usually not part of a strategy, but of the implementation plan. In this case we might just want to mention that low costs attempts should be done to reach people and capture media attention.

### 3. Project Output

In this section you have to include what the project produces, not actions as such. A project does not produce as output “training of government employees” (this is an action); it produces “trained government employees”, the consequence of the action taken within the project.

The output will depend on the needs of the country, the script… and then on the strategy and on the specific plan that implements that strategy. You will have to make your own list, but possible outputs from the project are:

- An English/Local-Language glossary of computer terms. Between 1,200 and 2,500 terms. As a side product of this, a full Local-Language/Local-Language computer dictionary.

- **A set of programs translated to the local language**, with all their support materials - delivered for all platforms - that is sufficient for all the basic uses of computers (office and communications software).

- A full computer operating system and all the necessary office and entertainment applications needed by a normal computer user, and all of it entirely in local language. A user will only see the local language in his screen. The system will include **full documentation in local language**, in electronic and paper formats.

- **If the script is not a usual script for which many fonts exist**, a set of **10 to 20 computer** fonts (Unicode OpenType fonts) to be used in the application menus, for word-processing or for computer design.

- If no standard keyboard that has all your language needs exists, **A keyboard layout**, supporting drivers and a few thousand physical keyboards that support the above mentioned Unicode fonts.

- **An installation disk** that —including all the above software and documentation- is very easy to install. Manufacturing of a few thousand copies of this installation disk, 1,000 of them accompanied by full printed documentation.

- **Training materials addressed to end-users**, including the use of the system, of the applications and a training course for typing with the new keyboard (if applicable). **Training for trainers** course and materials.

- **Computer end-user trainers trained** to teach the new applications using the above mentioned materials.
• University professors, students and software development company personnel trained on advanced Linux and Open Source software and application development using the Local Language script support tools provided by the project. Computer vendor personnel trained on the installation and parametrisation of the system.

• Open Source Expertise Centers in Government and/or Universities, including trained professors and students in equipped facilities with reasonable Internet connections.

• Software development companies empowered to develop applications based on Open Source Software which require support for Local language script.

• Government personnel trained to use the system.

• Government personnel expertise on software purchasing, coordination of applications between similar administrations and analysis of priority applications for improved governance.

• Marketing materials for deployment of the system.

• Widespread publicity of the system either directly or through computer and software vendors.

4. Benefits and beneficiaries of the project

A possible way of developing this section is to first talk about abstract benefits, and then talk about the specific groups that benefit from these developments.

The generic benefits include:

• Effective separation of the foreign-language skill from the computer-use skill. English will no longer be a requirement for learning the use of computers.

• Very significant reduction of training time needed for the use of computers, especially for the less favored, who before had to learn English (or some other foreign language) first. A job attained thanks to having one of the skills (computer use) can later provide the necessary economic resources needed to study the other skill (English), instead of needing to be economically dependent while both skill are acquired.

• Allowing the country to enter the world of technology without forfeiting its culture. Avoiding the entrance of an excessive number of foreign words in the local language. Turning the computer into a tool that reinforces local culture, by clearly defining and permitting the right ordering of words, the right format for dates, as well as a number of other cultural issues that computers in a foreign language could change irreversibly, adopting foreign forms.

• Allow people in the country to communicate through e-mail in their local language.

• Opening the way for the development of computer systems for the National Government that will allow Government employees to work entirely in their language and manage databases of local names and data. The same applies to utility companies (electricity, water, phone, etc.) which will
develop local language databases, reducing costs and giving better service to the citizens. Open the door to computerised decentralisation of the Government.

- Creating the possibility of **reaching out with computers to schools, remote provinces and governmental offices**, where, because of age, resources, or plain lack of need for higher education, English is not a common skill. All of them places where people are eager learn - and need to learn - how to use computers, to be more productive as well as to have easier access to better paid jobs.

- Allow effective **computerisation of SMEs** and improve taxation fairness and transparency.

- Empowerment of local software development companies to work for the local Governments and private companies in local language, a skill that few others outside the country will have.

**Beneficiaries of the project include all Citizens**, who will have easier access to computer technology –while reinforcing their culture- and who will benefit from **better governance** through the use of computers. They less favored economically will benefit the most, by having easier access to jobs that require the use of computers (but not the English language) which before were inaccessible to them.

**SMEs** will benefit of better management through the use of computers.

**The Government** will be able to computerize its different departments without incurring in large training expenses, and will avoid havings its employees blocked long time with training (English + computer). The possibility of computerising the Government in its own language (especially in the countries in which the script was not correctly used by computer system before) is the most important advantage, with a major impact on being able to create databases that use names and data in local script, something that still does not happen in many countries and regions.

### 5. State of the technology to be used

#### 5.1. Open Source Software

In this part we attempt to briefly explain what Open Source is, without entering the technical domain. For example:

The last decade has seen a phenomenon that in the preceding one would have been thought as impossible. A community of volunteer computer scientists has put together and maintains a computer operating system, advanced user interfaces (desktops) and a number of applications (including office software, such as word processing or spreadsheets, and Internet tools) that compete in quality, appearance and robustness with some of the most developed proprietary operating systems (such as Microsoft Windows), producing – in some cases – better and safer applications (as it is the case of browsing and e-mail tools). All this software - grouped under the name of Free and Open Source Software (FOSS) - is in the public domain and can be used or translated by anybody, without having to pay any licenses. Large computer companies such as IBM or Sun Microsystems support the FOSS model and develop and distribute Free and Open Source software.
A larger and larger number of Asia and European national and local administrations have developed or are developing policies to use Free and Open Source software instead of proprietary systems and tools. These policies are also being supported and pushed by bodies such as the United Nations Development Program.

Some FOSS user interfaces and applications are now being translated into different languages, including some that use complex scripts, such as Khmer, Thai or Indian languages.

Specifically, Free and Open Source software opens the door to creating locale language software in countries in which large companies like Microsoft are not interested because they do not consider these countries as profitable markets.

In parallel to Open Source, the idea of Open Standards has also grown as a way to assure that a company's or Government's data is not tied to a specific software vendor, and the customer can change supplier or vendor without loosing its data. Open Standards permit any software provider to adapt to this format, opening the choices of supplier for Governments and corporations. A growing number of Governments are mandating the use of Open Formats (such as OpenDocument format for office applications) as the format for exchange and storage of data within the Government.

5.2. Encoding for the target language

Here is where you really need to start doing research.

As computers are concerned, there exist four types of scripts:

1. Latin scripts and derivatives (Spanish, French, English, German, Swahili, Tetum, Vietnamese, Samoan, etc.)
2. Languages that have their own script, but still characters are written one after the other from left to right, as in the Latin Script (Cyrillic, Greek, etc.).
3. Complex Text Layout (CTL) scripts: Indian and south East Asian languages (originated in the Brahmi script), as well as languages that are written from right to left. Includes, Hindi, Kannada, Malayalam, Khmer, Thai, Arab, Urdu, Hebrew, etc.
4. Chinese, Japanese and Korean (and old Vietnamese, no longer used). These languages are usually referred to as CJK (or sometimes as CJKV).

For all languages, except some CTL languages, there are character set that can be used by the computer to represent that language. For MS Windows, the most common character sets are:

<table>
<thead>
<tr>
<th>Name of charset</th>
<th>ISO code</th>
<th>MS codepage</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI_CHARSET</td>
<td>ISO-8859-1</td>
<td>CP1252</td>
<td>(Most Latin scripts)</td>
</tr>
<tr>
<td>BALTIC_CHARSET</td>
<td></td>
<td>CP1257</td>
<td>(Baltic republics)</td>
</tr>
<tr>
<td>CHINESEBIG5_CHARSET</td>
<td></td>
<td>CP950</td>
<td>(Chinese)</td>
</tr>
<tr>
<td>EASTEUROPE_CHARSET</td>
<td></td>
<td>CP1250</td>
<td>(Georgian, Armenian?)</td>
</tr>
<tr>
<td>GB2312_CHARSET</td>
<td></td>
<td>CP936</td>
<td></td>
</tr>
<tr>
<td>GREEK_CHARSET</td>
<td></td>
<td>CP1253</td>
<td>(Greek)</td>
</tr>
</tbody>
</table>
Nevertheless, these character sets are giving way, little by little, to the Unicode standard.

Unicode is a single, very large, character set that contains all major scripts of the world, and pretends to end up including all of them. The Unicode standard is being developed by the Unicode consortium, a consortium of large industry players that affect the way in which language technology is implemented in computers (IBM, Sun, Microsoft, Adobe, etc.).

Unicode is supported by the last versions of MS Windows (2000 and XP), as well as by most Linux based language support libraries. Many of them ONLY support Unicode, and not other characters sets. Unicode is clearly the future, and the other character sets are probably bound to disappear with time.

Even if they are encoded in Unicode, some languages require specific display logic, and they are not yet correctly displayed by MS Windows or by some of the Linux-based platforms. Work is being done to improve this.

5.3. Script and Language support in different platforms

In this part you should specify if your script is supported by the different platforms, or if any work related to this needs to be done.

For languages that use the Latin script, support for the display and printing of the script in different platforms is implemented, so localisation to these languages does not require developing script support, but for many others it is necessary, especially for East Asian languages (Chinese, Japanese, and Korean), Indic languages, right-to-left languages…

Windows

On the Windows platform (Windows 200 and Windows XP) support is available for most of the existing scripts in Unicode, even if it is not perfect for all of them. Support for Unicode scripts is given through the Uniscribe Engine, contained in the file c:\windows\system32\usp10.dll. The version of this file that is included in Windows XP (and also in Windows XP SP2) is quite old. A newer version is shipped with MS Office 2003, but it is placed in a directory that is only used by MS Office. In order to assure that this file is used by all applications, it needs to be placed in the system32 directory (not an easy task, as the system protects itself from changes). Information about which languages are supported by Uniscribe should be searched for at the Microsoft website, in the
typography section: http://www.microsoft.com/typography/specs/default.htm. If support for your script is not there, you should plan to contact Microsoft directly.

**Linux**

Linux by itself (the Linux Kernel) does not support (nor needs to support) any scripts, only basic ASCII (for languages that use the Latin script characters used in English, and no more). Script support is done at a higher level. For some languages this support resides at the X.11 layer, but for complex text layout scripts it is necessary to develop support for the script at the user interface level (include the support in the libraries that are used by the user interface).

In order to have a wide support in different interfaces and applications, it is necessary to develop (if it not there) support for your script in three different libraries:

- **ICU.** (International Components for Unicode), a library produced by IBM that is used by Java (in several platforms, including Windows and Linux), by PHP, and by OpenOffice on Linux (on Windows platform OpenOffice uses Uniscribe). Even if most of the library is developed and maintained from IBM, it is Open Source and it accepts code from outside sources for support of other languages.

- **Pango.** This library gives support to the Gnome user interface and to many applications in the Gnome environment. It is also used by Mozilla products (special versions) for support of Indic languages.

- **Qt.** Library that gives script support to the KDE user interface and applications in its environment.

If support for your script is not available in these libraries, you should include in your project the development of such support.

Script support has several layers. The basic one is Layout support, with allows the applications to display your script. Higher level support makes reference to how the cursor should move inside the text (especially important if the language is written in “clusters”) and to line-breaking.

5.4. Locale and collation data

There exists a common repository (a central location for storing information) for cultural data used by computer programs, which includes data from all languages and cultures in the world (one by one) that needs to be used in computer. This includes information on date formats used in each country, number formats (do you use the “,” or the “.” as a decimal separator, etc.). This data is stored in XML format in:


If data for your language/region does not exist, you should seriously consider gathering it and sending it. Look at the www.unicode.org website for details.

Part of the cultural data is the alphabetical order that is normally used in your language. Even if the order in which the characters are in Unicode is sometimes close to the order that it is traditionally used in each country, it is really the exact one; collating (alphabetical sorting) order has to be specified.
This is slightly more complicated, because besides the Unicode code-point order (forget about this order), there is a specific list in which all Unicode characters are sorted:

http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt

If you agree with the way in which the elements for your language are sorted in this list, you do not need to do anything. But if you disagree, you need to prepare and send specific collation data about the changes that need to be done in this list for your collation to be correct. More information about this is available in:

http://www.unicode.org/reports/tr35/#%3Ccollations%3E

Again, for your localisation project, at this point, you should only consider if locale and collation data is already available in the repository, and if any work on this should take place. There are at present a few other repositories of locale data to which you might need to send locale files (in different formats) if you localise the applications or user interfaces: gclib (for C compilers), OpenOffice locale, KDE locale, Debian locale, etc...

5.5. Fonts

In this section you should indicate if complete fonts are already available for you script, or if – otherwise – it is necessary to develop them. This will not be the case if your language uses the Latin script or other common scripts that already have a large amount of fonts developed. It will only be necessary if the script used by the language is fairly new to computers and no good Unicode fonts have been developed.

It is important to understand that a font has several characteristics that determine its adequacy. One of them is the encoding scheme that it follows: legacy encoding (arbitrary encoding of characters in one byte), a specific national encoding standard or the Unicode standard (assigning to each character the code established in Unicode). The fonts that you will need for this project are fonts that follow the Unicode standard.

5.6. Translation (glossary)

On the language side, before starting localisation, it is necessary to define the terms that will be used for user interface terminology in local language (how to say “file”, “folder”, etc. in local language). These terms need to be compiled in an English/Local-language computer dictionary. A basic dictionary has about 1,500 terms, while a dictionary that would allow translating an advanced application such as OpenOffice requires at least 2,500 terms, including words used in accounting, mathematics and statistics.

The project should state if such glossary exist or if it needs to be developed. If so, it should always be the first part of the project, as translation cannot start until a basic glossary is finished.

6. Localisation work that needs to be done (short descriptions)

This chapter should have clear non-technical short descriptions of the tasks that need to be developed for the project, and which will be included in Annex A. They can follow the same structure as the Annex, or a different one that is easier to understand for less technical readers.
7. **Government support**

A country moves through its Government. Being able to enlist the Government as a partner for the project completely changes the perspective, the scope and the possibility for impact. In this sort of project, the Government can be involved in four different ways:

- Taking co-ownership of the project.
- Writing policy that will support the use of Free and Open Source, of local language software and of Open Standards.
- Migrating its computers to use local language or Free and Open Source software.
- Using its teacher-training structures to teach the use of local language software and to create expertise on the use of Free and Open Source software.
- Having a FOSS expertise centre that provides training and networking to Government, high-level training institutions, computer vendors, development companies and anybody else that needs to acquire Linux expertise.

7.1. **Policy**

Policy can take quite a number of forms:

- Development of high level policy in which FOSS and Open Standards are encouraged and must be included in lower level policies such as Government appropriation policy.
- Specifying a Master Plan for the re-computerisation of the country in its own language, including migration of the Government and teaching of this software in the country's schools.
- Purchasing policy in which Open Source must be considered, and that assure that no de-facto monopolies can be created by proprietary software companies.
- Adoption of Open Standards for data exchange and storage inside the Government.
- Assuring that teacher trainers are formed to the teaching of Open Source local language software, and that these teachers teach in all of the Government's teacher training facilities.

7.2. **Access to Government migration**

The most important issue here would be the development of a Migration plan for Government offices. Other lesser actions include active promotion of the localised software to Government offices and promoting or enforcing the use of Open Standards.

7.3. **Access to Government's training centres.**

This would be the implementation part of the policy leading to project the use of local language software to schools all over the country, using the teacher training centres of the corresponding ministry (such as education) as the appropriate tool.
8. Distribution and Deployment

The point at which most localisation projects fail is distribution. The translations are done with more or less quality. Linux based CDs are sometimes printed and somehow distributed... but that is where it all ends. There is no strategy, no follow up, no support, no understanding of how the market works... and it leads to these CDs never being used.

8.1. Support

In many developing countries it is easy to get the impression that support for proprietary software does not exist, but this is not really true. Even if there is no formal support from the company that manufactures the software, you can pull from the expertise of vendors, training places and even friends and work-mates who already know how to use it. A support structure is necessary for the deployment of any type of software. Ideally, you should try to use an existing structure, mainly computer vendors, training institutions and individuals. Once these players have the basic knowledge, support will be assured.

As support is dependent on present distribution channels, it is very important that the strategy that you decide does not confront such channels. A sudden change to Linux - up front - would probably be rejected by vendors and trainers, as they do not only need to gain knowledge of the applications, they also have to learn a complete new systems. It is always easier to go step by step, allowing these support structures to build up and to maintain an equilibrium. Otherwise they will reject your project.

8.2. Vendors

Ideally, the best way to distribute your applications is to assure that they are pre-installed with every new computer that is sold by a vendor (this is the strategy that has made Microsoft what it is). If the added value of your product is large enough, they will pre-install it in order to give more value to their customers.

Hardware vendor are a much better channel that software vendors, as the latter will not make much money on your FOSS software. Software distributors are a good channel in countries that do not accept or enforce software licenses. FOSS is the most legal product that they will sell, and the price and profit for them will probably be higher, but not much. It is nevertheless important to ensure that the software is available in their shops.

The PCs that can not be reached through vendors are those that are already installed. For these it is necessary to do installation campaigns or installation fests in which some specific people (or students in general) will be sent out to the country to install computers in organisations, government offices, cybercafes, NGOs, and any other computer they can get their hands on. This is the type of project that can enlist a lot of community support.

8.3. Training for end-user trainers.

Training is the key to usage. It eliminates the fear that users might have of approaching the applications and using them.

It is important to penetrate the existing training industry. Present teachers have teaching jobs (teaching proprietary products) that can be turned into jobs that teach FOSS products. Their employers are given the opportunity to change by having trained teachers. Training new computer teachers (not yet employed as
such) does not assure that you are creating training opportunities for students. This does not mean that they should not be taught, but that they should have a lower priority.

Training is the vehicle for the computer terminology that has been developed. Offering the training industry free good standardised training materials reduces the barrier for them to start teaching your applications. The development of a certification system also helps them adding value to their teaching, as their students will receive an official certificate, not given by some training place, but by an established certifier.

Free training for trainers is the key to penetrating the end-user computer training sector. The training for trainers should be directed at having them teach your applications by using the materials that you have developed. What you are actually teaching them is not the application itself. You teach how to use the standard training materials.

You should plan here for the development of training materials (about 70 hours for applications, and 30 more for the Linux user interface (basic and advanced). You should also develop training-for-trainer materials, and should consider the creation of a certification system or he adherence to an international one, such as the International Computer Driver's License (ICDL). The international one do have a cost that might be too high for your country, but you can create your own program following their curriculum.

8.4. Deployment on Windows platform

The two most important issues that prevent a user from changing to a new computer application are fear of losing productivity and facing a program that they do not understand, and fear of installing something new in their computer and “breaking” it.

These two “barriers to change” are eliminated by assuring that the applications are installed in all possible computers an by providing training.

8.4.1. Installation campaign

Installation for both installed systems and new systems should be considered. This requires negotiation with computer vendors, so that they will pre-install the applications, and an active campaign to reach as many [already installed] computers as possible, in the Government as well as in civil society. It is advisable to have specific human resources (a high technical level is not required) dedicated to installation, traveling through the country to assure that Government offices, NGOs, cyberspaces and private companies can have the applications installed in their computers. Support for vendors is also necessary, with identified resources (of a higher level in this case) dedicating some of their time to resolve any doubts coming from computer vendors.

8.4.2. Training campaign

The training campaign tries to assure that the present computer training institutions in the country that cater to the needs of final users actually change to teaching Free Open Source applications in the local language.

The training campaign requires that training materials are available and free, and therefore cannot be started until the materials are finished and
tested. These standard materials create standardisation of the training market, and assure that the terminology that you have developed is passed to the students.

The campaign must be directed to computer teachers that are already teaching, as they are the ones that have the opportunity of training students, converting also their institutions (in Government, University, public schools or private schools) to the teaching of Open Source local language applications.

A reasonable campaign requires at least four full-time teachers employed from the time the materials are finished until the end of the project. They can also be employed before and participate in the writing of the training materials.

Also, some funding should be considered to help (push) training institutions start their new programs with Open Source software.

A training campaign can also benefit enormously from a Certification program that certifies teachers and students as capacitated to teach/use the Open Source tools. The value is much higher if the program is supported by the Government, issuing official certificates. Such program requires specific resources.

8.5. Deployment of Linux Platform

A serious deployment of applications on Linux platform requires a country that is technically prepared to support Linux. If the distribution channels (computer vendors and training institutions) are not prepared, the project will definitely fail. For deployment to have any possibility of success, the preparation requires: 1) A technical base capable of supporting Linux (a sufficient number of Linux Administrators working in the right places) and; 2) Users already working on the FOSS applications that they will find on Linux (OpenOffice, etc.), with a lower level of dependency on proprietary tools. These two factors lead to non-traumatic change.

8.5.1. Installation campaign

The Linux installation campaign is a continuation of the campaign to install applications on Windows platform. The same team, already used to deal with technicians, vendors, etc, will be the one to lead the work of migrating computers to Linux and assuring that vendors pre-install it (in many cases as part of a dual-boot installation, together with a proprietary system). The same team should be considered until the end of the project.

8.5.2. Training campaign

The training campaign has two components:

1) Technician training. Using training materials in the local language, a sufficient number of Linux Administrators should be trained (about two to five weeks of training for computer scientists with a bachelor's level). The work to be done includes the development/translation of the training materials, the preparation of teachers and the actual teaching.

2) Teacher training. End-user computer teachers will be taught how to use a graphical user interface on Linux, how to install Linux, how to teach the
use of the Linux user interface and how to use the applications under Linux. The training will be directed to the same teachers that already followed the courses on the application, as it would be useless if the applications are not known. The same team that before taught the use of the applications on Windows will continue its work to teach the use of Linux and the applications under Linux.

9. Development of Free and Open Source Expertise

As mentioned above, support is the fundamental base for deployment. The support structure must include all partners in the project, including Government, vendors, universities, private development software companies and NGOs. It is the responsibility of the project to create the necessary training structures to assure that all these partners acquire the required expertise.

The expertise must cover specially the work that needs to be done to deploy the applications, and at a much deeper level, around the Linux operating system, as most system and networking expertise must be assumed to be on proprietary systems. The goal is to reach a point in which the choice of proprietary or Open Source operating system does not carry support issues. Technical support services must handle both equally, depending on customer's choice.

9.1. Universities

Universities must assume the most technical part of the teaching, concentrating on the Linux operating system, concentrating know-how and creating the highest possible number of engineers trained to use Linux, as part of their standard curriculum. Their teachers must be trained and curriculums must be developed. Some standard curriculums¹ and free teaching materials² already exist. Training materials should be translated to (or developed in) local language.

9.2. Vendors

Expertise in computer vendor technical services is the key to deployment of FOSS operating systems. If these services are confident, they will willingly pre-install Linux - or dual-boot (Linux+Windows) operating systems - and maintain them. The strongest training effort must be directed at training their personnel, and to assure that they assume a role in the social development of the country, which at the end will come back to them in the form of a more developed market in which – by having new customers enter the market – will grow at an accelerated speed.

Software vendors or distributors should be informed of the difference between the systems and the new software that is being deployed, so that they can also be a part of it.

9.3. Private companies

Development of a local software development market is fundamental for the Government to effectively change to use Free and Open Source operating systems. Expertise in private sector usually grows out of “need”. Engineers are trained when projects require certain knowledge, not before, unless the training was acquired at the university. The important thing is that the training is

¹ http://www.lpi.org
² http://www.iosn.net/training/system-administrator/
available when required, and that slowly a pool of knowledge comes into existence in the private software sector, so that developing under Linux environment will not be an issue after one or two years. You need to ensure that training, in a format convenient to working professionals, is available.

9.4. Government

In order to migrate in an orderly way, the Government required a centralised support structure, through which it distributes knowledge to all Ministries and local administrations.

These support structures, as well as how expertise will be distributed should be considered in the project.

9.5. Maintenance of localised Software

It might take about two years to translate all the necessary applications, including documentation, creation of training materials, etc.

After all this is finished, it is necessary to assure sustainability of the project by maintaining this software, meaning that when new versions come out, they should also be translated and distributed. The translation of new versions is not so complicated, and major packages of software tend to have a long life cycle (one year or 1 and half years). Smaller products might have a shorter cycle.

It is not necessary to constantly upgrade the software (except when security fixes are involved). The one and half year cycle seems to fit the user's needs, optimally, but the permanence of an installed program might last all of a computer's life without upgrading.

It is nevertheless important to assure that new versions become available, otherwise the original advantage produced by the language will be eventually lost.

The project has to consider who and with which funding will assure maintenance and sustainability of the software once this project is finished.

10. Project plan

10.1. Organisational structure for the project

If known, this section should mention what organisation(s) will implement the project, and if Government involvement is desired or expected. If possible, the qualifications of the organisation that will implement the project should be included. The description here should be short, it can be completed with an Annex that describes the organisation and its strengths in further detail.

10.2. Calendar

The calendar is probably the last part of the project that will be prepared, as it requires evaluating the amount of resources that each module of the project will take, and which will be the order in which these modules will take place, depending on which other jobs need to be finished before a specific one can start.

The calendar also depends on available human resources, and is probably the most difficult part of the project to prepare.
While preparing the project we have to look at all the task that will be required, and will have to assign resources so that no part is unnecessarily delayed. For example, if a task requires one translator for one year, and many other tasks cannot start until this one is finished, it might be wise to dedicate to it three or four translators, so that it can be finished in three or four months, unblocking tasks that come after it. Resource planning is difficult, and it should be done carefully and realistically.

10.3.Required Human resources

As with the calendar, it is not possible to establish the human resources that will be needed until the amount of work required is known (as well as the qualifications necessary for each task). The human resources that the project will use will determine the calendar that can be finally implemented.

Human resources do not need to be the same along the project. It is probably easier to start with a small team, consolidate a glossary, acquire knowledge on tools and methodology, and then start making the project grow, aware that at any new step new entrants will require some training at the beginning.

10.4.Milestones and evaluation

This should be prepared after the calendar. It should determine when the calendar will reach significant milestones, and how the project management or a possible donor can evaluate if the goals for that milestone have been reached.

Several tasks might lead to the same milestone. As an example, the localisation of several programs might need to be finished in order to reach the milestone of “having all basic applications for office and Internet localised”. You should plan carefully your human resources to ensure that specific tasks that require more work are assigned enough resources to finish as soon as possible, and not having one specific task delay a milestone, when all others are finished.

10.5.Risk factors

This section should seriously analyse what factors might endanger the outcome of the project or delay its calendar, and how serious these risks are. If should give a clear idea of how probable it is that the project will or will not be accomplished, and if problems are detected, how is it possible to correct them.

10.6.Future actions once the project is finished.

Here the writer should try to explain how the project will not just die once it is finished. It is interesting to think how it might continue and how it will be sustained (assuming that this project -as the first phase- is funded).

11.Budget

In order to develop the budget for a project, it is necessary to go into all its details.

The budget can be organised in two different ways: defining the resources that the project will have (and then structuring the work around these resources) or structured in modules (and for each module it will be necessary to define the required resources).

The modules need to be developed anyway, in order to figure out the amount of resources that they will require. Each module should be a specific task (such as
for example developing 10 fonts). For the module we should analyse the required human resources, the computers that will be used, where it will take place, and all other costs associated to the module. In our example, the module might take one typographer for one and half years using one computer.

When computing the cost of human resources, it is also necessary to include the cost associated to the employee, such as the proportional part of the cost of the office, the salaries of the managers, etc... These general costs should be distributed among all modules. It is also possible to consider the general costs aside, as a different part of the budget.

The budget should contemplate the whole project, but it should also be divided, considering the each year separately, for funding purposes. Also, the cost of each module should be detailed, for transparency as well as to allow possible donors fund only specific modules, in case they do not want, or can, fund the whole project.

The budget can only be done after all the modules have been clearly defined.
Technical Annexes

12. Project modules

It is not necessary to go into deep technical details for each one of the modules that need to be accomplished, but it is necessary to give enough information to evaluate the amount of work necessary, and what each module will accomplish, and the type of resources that will be needed. It is also a good idea to specify its duration and its relation to other modules, such as if some module needs to be finished before this one can start, or if some task cannot begin until this module is finished.

Modules can involve technical work (translation, development), documentation, development of training materials, or actions, such as an installation campaign or a training campaign.

The use of specific project planning software can be very helpful in the preparation of the plan.

12.1. Language adaptation – Glossary

The translation effort must be done in an orderly way, preceded by the development of a Glossary of computer terms used in the user interface and in the applications. Once the Glossary is developed, all translators will use the same terms when translating software and user manuals.

A first minimal Glossary should have about 1,500 terms. List of English words that should be translated are available³. This tasks takes about 4 translator/months (2 translators for 2 months), and should be validated afterwards by computer science professors and linguists. A more complete glossary that covers most of the needs of a large application such as OpenOffice has about 2,500 words. The list of words is also available⁴.

The expertise required in the translators is a very clear understanding of the computer concepts in English, and being able to study the origin of the terms in English. A good knowledge of the local language is necessary, but experience on translation would be a secondary in this case.

The use of a glossary lead to requiring less prepared translators, and to be able to distribute the work between several translators, while maintaining the coherence and unity of the translation.

12.2. Language support

12.2.1. International standards

Most modern computer systems are based in the Unicode international character encoding standard. It is difficult to think about non-standard systems that will last for a long time. An action on this should only be necessary if the script used by the local language has not been included in the Unicode standard, in which

³ http://prdownloads.sourceforge.net/khmer/glossary-1567Terms-Empty-V2.ods?download
⁴ http://prdownloads.sourceforge.net/khmer/glossary-BySubject-Empty-2551Terms-V3.1.ods?download
Preparing a FOSS Localisation and Deployment Project

Localisation Toolkit

case you should start the (long) process to include it\(^5\). Only after the script is encoded in Unicode is it possible to start developing fonts and script support. All national scripts, and quite a large number of regional ones are already included in Unicode.

12.2.2. Fonts

If no fonts are available for your script, or if the fonts are not proportional\(^6\), you will need to study and develop fonts for the script. It is important to understand the different printing styles traditionally used in the language, and develop fonts for all of them, otherwise the user will keep reverting to other methods of printing. The first font to be developed will probably take some time, specially for complex scripts, but others should not take so much time. One or two of the fonts should be “hinted”, a process that makes the fonts very clear in the screen in small sizes. An effort of 18 months by a typographer should be considered, if fonts are needed.

12.2.3. Script support in software

The complexity of this work depends on the type of script. Few scripts are not yet supported, but if this is the case, a high level developer should be considered for the work, for two or three months of work. He should work in the implementation of support for the script in the ICU, Pango and Qt libraries.

12.3. Keyboard

In case no keyboard for the script (using Unicode) is available, then the drivers need to be developed, for both the Windows and Linux platforms. Depending on the computers present in the country, a Mac keyboard could also be considered. For complex scripts, a keyboard that uses the IIIMF technology should also be considered (but it is not a priority)

12.4. Cultural Data (locale)

Part of the localisation process is to assure that the computer knows how to use data in the way it is normally handled in your country. This includes date formats, number formats (does it use a decimal point or a decimal coma?), calendar, separators, etc. All this information, usually different for each culture, is called Locale Data.

There is a tendency to centralise all the Locale information for all countries in one single place in the Internet, to be used by all applications. This central storage place of locale data is maintained by the Unicode Consortium, and it is called the Central Locale Data Repository (CLDR). Data from this central location is more and more used by different applications and the authoritative source of data about how different cultures interpret data. There are still a few other key locations that still use their own source of data, these are the glibc library (for programs in C and C++ languages), OpenOffice locale, KDE locale, Debian locale...

---

5 http://www.unicode.org

6 Non-proportional fonts are those that include Latin and some other script, and in which the sizes for the two scripts are not compatible. Normally in these non-proportional fonts the Latin script follow the standard metrics (clearly readable at size 10), but the other script, in order to be equally read, needs to be displayed at double the size (20 points minimum). This has been the case for many fonts for CTL scripts. These fonts can not be used for localisation, as quite often the characters are intermixed, and must have compatible size for the same number of points.
It is important to make sure that locale files exist for these three locations. The CLDR and OpenOffice locales are both in XML format and not too different (the OpenOffice locale will tend to disappear in favor of the CLDR locale in future versions of OpenOffice). The gclib locale is written in C language.

About two weeks of work should be considered to find the necessary cultural information, prepare the locales, submit them and do the follow up until they are finally accepted.

12.5. Localisation of Applications

Localisation of applications is the most extensive and resource-intensive part of the work. Depending on the size and complexity of the application to be translated, the effort might be from a few days of a translator to several translation year (as in the case of OpenOffice). Technical complexity must also be considered; some applications do not need any changes in the code when a new language is introduced, while other require a number of changes and patches that the localiser must provide.

In order to evaluate correctly the amount of work required to localise these applications, it is first necessary to decide which applications need to be translated, and then search to see how many messages need to be translated for those applications.

The tendency should be to translate multi-platform applications that can be used in both Windows and Linux. Completion of this section should give as a result a list of applications that need to be translated and a total number of messages included in those applications (which need to be translated). This number will be later used to know the number of translators needed for the project and the duration of their work.

There are four major groups of applications.

- Office software
- Internet Software
- Multimedia applications
- Utilities

As office applications are concerned, the only FOSS suite that can really compete with proprietary software is OpenOffice. OpenOffice has a total of 57,000 messages to be translated, of which 24,000 correspond to the code and the rest to the help system.

In the realm of Internet software, the most usual applications that are translated are the two new products of the Mozilla family: Firefox (internet browser) and Thunderbird (mail client). For the localisation, if the user wishes to use the names of “Firefox” and “Thunderbird”, both trademarks of the Mozilla Foundation, he/she has to follow the strict, cumbersome and highly technical rules of the Foundation, but it is also possible to translate the applications in a much simpler way if a different name is used. Both applications have some common messages, the total for both being of about 7,000 to 8,000.

It is also interesting to translate at least one of the web-mail systems available (IMP, Squirrelmail, etc...) with about 1,000 messages, some instant messaging application, such as Gaim or chatzilla (1,000 more) and public proprietary
search engines or web-mail systems that are often used (Yahoo, Google, Gmail, etc.). A pool of 2,000 to 3,000 additional messages should be considered for these.

In the multimedia section you should consider applications for playing music and DVDs (such as VLC, 2,000 messages) and probably some image editor, such as GIMP (6,000 messages). This group can be increased with vectorial editors (Scribus will soon be multi-platform) and page-setting software (such as Inkspace).

Utilities should include software such as compressors/decompressors of ZIP and other compressed files (such as 7-zip), CD/DVD recording software, and other utilities often used by final users.

As a reference, a translator can translate - depending on experience and ability - 2,500 to 3,000 messages per month, and review as many in the same period. Counting on a 20% overhead is never a bad idea, when preparing the budget.

12.5.1. Software adaptation

Adaptation of the source of programs to assure that they handle a language or script correctly is better and better documented, and does not require a high level of knowledge in programming (for most projects). There are – nevertheless – other projects that still do not fully understand the difference between programmers and localisers and unnecessarily complicate their localisation systems with technical steps. The different communities tend to be very helpful on the work. Some human resources need to be considered in order to follow the technical details of the different localisations. The people assigned do not require prior experience in Open Source, but they will have to interact with the communities and be prepared to learn the specifics of each project.

12.5.2. Translation

Translation, once the habit of using translation editors is acquired (a few hours, requires very little technical help. New projects are being developed to make this part of the work easier. As this document is concerned, only the amount of translation (as described above) needs to be defined, depending on the different pieces of software being considered.

12.5.3. Documentation

It is not possible to hope that a community will start using some software if there is no documentation on how to use, either in the shape of help files, on-line help systems or documentation, in paper and/or in electronic format. Documentation for all applications localised should be carefully considered. It can be prepared with - or as a side-product of - the training materials, as these training materials force the creation of a curriculum that can be later developed in text as a user manual.

12.5.4. Development of training materials

The development of training materials is key for a project to reach the training industry. This group is one of the two most important channels that drive a user to decide which software he or she will use (the other one being hardware/software vendors). The development of training materials
Localisation Toolkit

Preparing a FOSS Localisation and Deployment Project

tends to take more or less the same amount of resources than translating a given program. Developing standard and free training materials simplifies the change of these institutions teaching proprietary software to teach Open Source software, and assures standard high quality teaching.

12.6 Localisation of a Linux User Interface

The same technology used to translate the applications is used to translate one (or both) of the Graphic user interfaces that can be used on Linux or other Free Operating systems: KDE and Gnome. The translation of a user interface completes the work necessary to have a complete computer system that is entirely in a given language.

12.6.1 Translation

A basic effort of translating about 20,000 messages should be considered. Each user interface has a number of applications that are well integrated into it, and these can also be translated, even if sometimes cover uses of the computer that are already solved by other applications that you might have already translated. A maximum number of messages to be considered would be around 50,000, and this would include a good number of new applications (utilities, games, multimedia, etc...) and might in many cases be the same applications that have already been considered in the “applications” section above.

12.6.2 Documentation

As with applications, user and reference documentation in local language should be developed (probably after finishing the training materials).

12.6.3 Development of training materials

Learning how to use the user interface is also part of the basic training for new or migrating users. The development of training materials will definitely facilitate training institutions to start training. About three months of a person should be considered (after the software is localised).

12.7 Packaging

12.7.1 Linux-based [and Windows-based] distributions

Effective software distribution in developing countries can only happen through direct distribution (CD's, etc), and ideally through pre-installation by computer vendors.

For distribution on Windows platform, the preparation of a CD with the software is simple.

The preparation of a Linux distribution is more complex. It requires either assuring that all your translations are included in a general distribution (such as Debian, Red Hat or Suse) or developing your own distribution for local consumption, simpler for hardware vendors, but much more complicated for you to prepare, as it requires deeper knowledge about the Linux operating system, installation, etc... It would be interesting to schedule a couple of months of work by a Linux expert with experience on distributions. In order to have a real distribution working in the local language, the installation program of the distribution must also be localised.
12.7.2. Installation documentation.

Hardware and software vendors, as well as the training industry and final users, will need as detailed as possible instructions on how to install the FOSS applications being developed under this project. Specific documentation should be developed, to reduce the amount of support that will be required later.

This also apply to installation instructions in Khmer for the chosen Linux distribution.

At least a month of work should be considered.

13. Other Informational Annexes

13.1. About the organisation that wishes to conduct the project

This section should describe -in a convincing way - the institution that writes and wishes to undertake the project. Making sure that the reader understand all the strengths and experience that make the institution be prepared to undertake the project and assure its outcome.

13.2. Curriculum Vitae of project managers.

This section also tries to reassure the reader of the fact that the person designated by the organisation to lead the project is prepared to run it from both its management and technical points of view.
How to do it

The order in which it is necessary to collect and prepare the information for a project is quite different to the order in which the information needs to be presented for a clear understanding from the reader.

Before starting to write, it is important to define what is going to be done, what will be the strategy, what are the tasks that will need to be implemented and what will be the milestones (and what their meaning will be).

You need to familiarize you with the Open Source projects that you are going to include in the project, get a clear understanding of what they do and what value they bring in and what is the cost of doing them.

Once the tasks are defined, you need to calculate what are the resources that will be required for each task: human resources, including their qualifications (in persons per month); computers, office space, etc... You can do a preliminary estimate of how long it would take, assigning a number of people to work on this task. For example, if localisation of OpenOffice takes one person/year, you might preliminary assign three translators and assume that the work will be done in four months.

The most difficult part is putting the task into one single plan.

- Each task requires other tasks to be terminated before this one can start. It also has other tasks that cannot start until this one is finished.
- You might have a limited amount of staff or volunteers available, so you will not have enough people to dedicate to each task as you would like. You will have to rework your tasks and recalculate their duration depending on available people.
- You might have a task that takes too long, unnecessarily delaying a milestone. You might want to switch some resources to work on this task, so that your milestone can be advanced.
- Many other nuisances that require rescheduling and reshaping of tasks.

Once you have clear tasks and a plan, you need to prepare the budget. It is probably easier to first calculate the budget of the whole project, independently of modules. You need to think of salaries (and their increment every year), computers, office space, office materials, connectivity, holidays, maintenance... You should make it as complete as possible. In case you doubt on include something or not, do include it, it is much easier to take things out later than to put them in. The budget need to be made for each year of the project.

Once you have a clear budget and of what things cost, you enter a new nightmare. You need to budget each module (the sum of the modules being equal to your total budget). This is done so that, possible funder who do not want or cannot fund the whole project, can see specific items (tasks) that they can fund, with a price-tag on them. They clearly see what task they can fund and how much does it cost doing it. Remember that the resources used in each task also must include a proportional part of the project manager's expenses (and other central expenses), as these cannot be budgeted separately when you budget module by module; otherwise you can find yourself with a project in which donors are funding modules... but you cannot afford to have an Internet connexion or a project manager.
Preparing a FOSS Localisation and Deployment Project

Localisation Toolkit

Once you have a list of tasks, their duration and their cost... then you can start writing the project in the order stated above (or your own order). You might at some point change things that you had done before (tasks, plan), but now it will be easy, as you will have a clear understanding of what each change implies.

During the whole process you must have very clear why you are writing this project. I believe that it has to provide the following:

- A clear plan that you can try to follow in the localisation process.
- An idea of what is necessary in order to do it, so that you can plan resources and monitor if you are advancing as scheduled.
- To the writer, a very clear understanding of the world of software localisation.
- A tool that may help you present the project to your organisation and to search for the necessary financial support.
References

This work is the result of the experience in the development and implementation of the KhmerOS project (www.khmeros.info).

This experience, together with the work of Elizabeth Da Conceição BAPTISTA, led to the TimorOS project (available at www.khmeros.info/download/TimorOS.pdf), a good example of the application of this methodology. The work on this project led us to improve and generalise the methodology. Another similar project is the SafariOS project for Uganda (at www.khmeros.info/download/SafariOS.pdf).

Other sources of good material that will help any localisation project are:

The Localisation Toolkit of www.iosn.net. The project is being developed by CDAC (India) and Simos Xenitellis and aims to give an insight on the technical aspects of localisation. Please note that the project has a similar name to that under which this document has been prepared (Open Source Localisation Toolkit), but it attempts to provide information on more technical issues.

Other interesting websites with information that might help any localisation project and process are:

www.wordforge.org (home of the WordForge project, aiming at facilitating the technical part of translation through computer-assisted software translation).

www.localisationdev.org (result of a process of brainstorming on localisation funded by the Open Society Institute).

This documentation has been written by Javier Solá as part of his work on an "Open Source Localization Toolkit" under a grant provided by the ICT R&D Grants Programme for Asia Pacific. The donors are APDIP (United Nations Development Program), the Internet Society, APNIC, Pan Asia Networking (IDRC), and Asia Media Information and Communication Centre. Copyright is shared by the Open Forum of Cambodia (grantee) and the donors, and is licensed under the Creative Commons license, allowing any use of its contents in any form. Attribution is kindly requested on “how-to” documents, but not on specific localisation projects developed under this methodology, even if they copy large parts of the contents.