Free/Open Source Software: Localization

Anousak Souphavanh and
Theppitak Karoonboonyanan

Asia-Pacific Development Information Programme
e-Primers on Free/Open Source Software

ELSEVIER
A division of
Reed Elsevier India Private Limited
17A/1, Lajpat Nagar IV,
New Delhi 110 024
Tel: 91-11-26447160
Fax: 91-11-26447156
Website: www.asiaelsevier.com

ISBN: 81-8147-754-5

Academic Press, Butterworth-Heinemann, Digital Press, Focal Press, Morgan Kaufmann, North Holland and Pergamon are the Science and Technology imprints of Elsevier.

Printed and bound in India

TABLE OF CONTENTS

FOREWORD

Free software means software that respects the user's freedom. It means that users are free to run the programs as they wish, free to study and change the software (or hire others to do it for them), free to redistribute copies to others, and free to publish modified versions. As a consequence, users are free to share, and form communities to exercise effective control over the software they use. Free software may also be gratis, zero price, but this is not always the case.

Some users refer to this as open source. That term focuses on the technical benefits that result when software can be reviewed by large numbers of developers and users, the business advantages of using it, and the business models that support its development and use. The term free software refers to the social and ethical importance of freedom, as well as to the practical benefits it brings.

The social and practical imperatives for sharing and changing useful works such as software create a revolution in what it means to publish. The technology of the printing press taught readers to think of written works as fixed – written once, for readers to use passively thereafter. This technology did not make it easy for readers, who generally did not have presses of their own, to adapt, improve, and share copies of books, and they became accustomed to these limitations. So when citizens in large numbers began to use personal computers, many did not question the legal and business systems that placed similar limitations on using software. But this time, the limitation was not a natural consequence of the technology. It was imposed by software developers, who found it profitable to keep users under their control. The developers forbid users to share the software, and by denying users the source code, prevent them from changing it.

Today, however, there is an alternative: free software, the product of a community built by a million willing developers – some volunteers, some paid to contribute. Free software allows users to take full advantage of what their computers can do. It is our door to escape from the limitations of the printing press, into a world where useful works, such as software, are developed among the users, by the users, for the users.

Where software is not made artificially scarce by misguided incentive systems that encourage development of software by locking up its users.

Where the users can collaborate as they see fit in making the software do what they want it to do. Where the users of software are free to run it, share it, and adapt it together or individually. This is the world of Free and Open Source Software (FOSS).

Non-free software keeps users divided and helpless. When a nation increases the use of non-free software, it is not development, it is permanent dependency. Only the use of FOSS permits sustainable development – it is technology that local people are free to learn about, maintain, adapt, and reapply.

How can governments move their countries towards FOSS? In two ways. First, switching to Free Software in schools will teach children the spirit of community cooperation, while producing graduates that are skilled in using and maintaining Free Software. Second, mandating migration to Free Software in government agencies will create demand for these graduates' skills, and build a local economy of Free Software support.

The International Open Source Network (IOSN) is an initiative of UNDP's Asia-Pacific Development Information Programme, and operates under the principle of "Software Freedom for All" (SFA). Its work includes provision of support and assistance, a centre of excellence, and an information clearing-house for Free and Open Source Software in the Asia-Pacific region.

Through the IOSN/SFA initiative, UNDP provides policy support and advisory services to government bodies, non-profit organizations, donor agencies and others. It publishes practical tools and materials, including simple “how to” primers and guidebooks, training materials, and live CDs of the GNU/Linux operating system for FOSS practitioners and end-users. It also supports FOSS R&D activities in localization and in other areas, and organizes conferences and training programmes to network stakeholders and strengthen local capacities. It welcomes both those interested in benefiting from these services and those who would like to collaborate in extending them.

I’m pleased to cooperate with the work of IOSN/SFA, APDIP and UNDP in taking the message of software freedom to the public and the development sector. Together we can help overcome the digital divide, and replace dependency with development.

Richard M. Stallman

ACKNOWLEDGEMENTS

The authors would like to acknowledge their debt of gratitude to the following:

Phet Sayo, the Programme Specialist for Building Capacities and Partnerships at UNDP-APDIP, for getting us started; Kenneth Wong, the Programme Consultant for the International Open Source Network, UNDP-APDIP; Tan Wooi Tong, the Programme Specialist for the International Open Source Network, UNDP-APDIP, for their patience and hard work. Managing and working on this large-scale project with them as our colleagues has made our lives less complicated. The team not only facilitated the project but also provided the feedback and input that makes this primer unique.

Sarmad Hussain, M Sasikumar and Javier Solá, the coordinator of KhmerOS, the Khmer (Cambodia) Software Initiative, for reviewing this primer and providing invaluable input. We want to specially thank Javier for his contribution on pages 12-15 on free/open source software and its benefits and the case study of Khmer localization.

Vorasone Dengkayaphichith, Vincent Berment, Alberto Escudero-Pascual, Pramod Raghavendra and Guntapalli Karunakar, who read and critiqued this primer.

Arthit-Thai OpenOffice team member, Indian and Vietnamese participants for their responses to the surveys.

Fred Williams, for his invaluable input, advice and suggestions.

PAN Localization Project (*www.panl10n.net*), PAN programme of IDRC and Center for Research in Urdu Language Processing (*CRULP, www.crulp.org*) at National University of Computer and Emerging Sciences (*www.nu.edu.pk*), for providing information from the survey during the "Fundamentals of Local Language Computing" training used in the second section.

Anousak dedicates this work to his family and his late parents.

INTRODUCTION

This primer provides a broad perspective on the localization of Free/Open Source Software (FOSS) for the benefit of policy- and decision-makers in developing countries. It highlights the benefits and strategies of FOSS localization, along with case studies from various countries that are on the road to software freedom.

The primer begins with an introduction to localization and the benefits of choosing FOSS over proprietary software. The next section provides a survey of initiatives and efforts in localization of FOSS within the Asia-Pacific region, including best practices and lessons learned specifically in countries such as Viet Nam, Thailand, Cambodia, India and Malaysia. The primer also provides three case studies of localization efforts in Thailand, Lao PDR and Cambodia, as well as recommendations on technical issues, resource allocation, skills and tools, implementation, costs and language considerations.

To help localizers get started, two annexes regarding key concepts and the technical aspects of localization are provided. These are intended for project managers and implementers who are planning software localization projects.

Computers, English, and Local Users

It is almost impossible to use a computer if you cannot read the instructions, buttons and menus. Thus it is not surprising that many countries in Asia lag behind Europe and America in the adoption and use of modern computer technologies at work, in schools, and in the home.

Computers don't have to display everything in English. In fact, the computer "doesn't care" what language is displayed, since everything is ultimately converted into ones and zeros. Even if the display is translated into another language, the computer continues to operate as before.

Translating software is nothing new. This process, which is known as 'localization', is not technically difficult. However, it requires professional management, a team of translators, and financial resources, especially for the initial translations.

Commercial companies have localized software for specific markets. Typically, they recoup their costs by charging license fees for the localized versions of their software. In countries where the average citizen cannot afford to pay the fees, they either do without localized software or resort to illegal copying.

All countries want the benefits of localized software, but some cannot afford the expensive licenses. With FOSS, this problem is solved. A combination of policies that encourage software localization/ translation into different languages and the ready availability of FOSS, presents an ideal means for expanding computer use worldwide.

Now is the time for developing countries to embrace the FOSS movement, and accelerate their adoption of computing technology by localizing FOSS.

Localization

Localization is the process of adapting, translating and customizing a product (software) for a specific market (for a specific locale or cultural conventions; the locale usually determines conventions such as sort order, keyboard layout, date, time, number and currency formats). In terms of software localization, this means the production of interfaces that are meaningful and comprehensible to local users.

The Localization Industry Standards Association (LISA) defines localization as: "Localization involves taking a product and making it linguistically and culturally appropriate to the target locale (country/region and language) where it will be used and sold."[1] Typically, this involves the translation of the user interface (the messages a program presents to users) to enable them to create documents and data, modify them, print them, send them by e-mail, etc.

Technically localizing FOSS is no different from localizing commercial software. Fonts must be changed, keyboard layouts devised, and standards adopted. The difference is price and licensing. With FOSS, the price is lower and the license open to all. For saving money and time, nurturing local innovation, and combating illegal copying of software, FOSS localization is a better alternative.

The Importance of Localization

Currently, people who want to use computers must first learn English. In a country with low literacy rates, this blocks access to information and communications technologies (ICTs), especially for the rural poor and women who do not have equal access to education. Even after having learnt English, users must pay hundreds of dollars to license foreign software, or resort to widespread illegal copying of software, in order to gain access to ICTs. In short, access to information technology is one of the keys to development, and localized FOSS applications remain a crucial missing link in communications infrastructure.

Localization brings the following benefits:

- Significantly reduces the amount of training necessary to empower end-users to use a computer system.
- Facilitates the introduction of computer technology in Small and Medium Enterprises (SMEs).
- Opens the way for the development of computer systems for a country's national, provincial and district level administration that will allow civil servants to work entirely in the local language and manage databases of local language names and data.
- Facilitates the decentralization of data at provincial and district levels. The same applies to utility companies (electricity, water, telephone), who will develop local language databases, thereby reducing costs and giving better service to citizens.
- Allows citizens to communicate through e-mail in their own language.
- Empowers local software development companies to work for the administration, the public sector and private companies.
- Provides the local design industry with good fonts.
- Helps universities train more software engineers.

The beneficiaries of this multi-stakeholder project are**:**

- Directly, all local computer users, who will have easier access to the use of computers as they will not have to learn English first.

- Indirectly, through improvements in governance using native computer systems, all local citizens in the quality of their dealings with the administration.

- The local government who will have the opportunity to develop databases and applications in the local language. Sufficient technology and empowered local development companies will be available. The government will also have the tool to coordinate applications among similar administrations (e.g., provinces), so that IT-based improvements in governance can be made at the lowest possible cost.

- The software industry. The government's use of standards-compliant computer technology encourages software companies to start developing compatible computer systems that will be used by the different bodies of the administration, thereby creating a stable software industry in the country. Once this expertise is developed (using FOSS), these companies will be empowered to undertake similar projects for foreign companies at extremely competitive prices, facilitating sales beyond the local market.

[1]*The Localization Industry Primer, LISA - The Localization Industry Standards Association*, 2nd Edition, 2003; available from *www.lisa.org/interact/LISAprimer.pdf.*

What is Free/Open Source Software?

The last decade witnessed a phenomenon which in the preceding one would have been thought of as impossible. A community of volunteer computer scientists has developed computer operating systems, advanced user interfaces (desktops), and a number of applications that compete in quality, appearance and robustness with some of the most advanced proprietary software (such as Microsoft Windows).

The term "free" in Free/Open Source Software refers to freedom to use, study, modify and share the software. The freedom to share FOSS implies that it can be used and translated by people without their having to pay any fees. However, some software that can be used without having to pay user fees, such as 'shareware' or 'freeware', cannot be studied, modified or shared, which means that they are not FOSS.

At one time, FOSS was exclusively developed by volunteer enthusiasts. Today, however, even large computer companies such as IBM and Sun Microsystems support and develop FOSS.

A growing number of European national and local administrations have developed or are developing policies to promote the use of FOSS instead of proprietary systems and tools. This not only gives them independence from commercial vendors, but also nurtures their own software development industries.

There is little doubt now that FOSS is viable and of high quality. Therefore, many governments are choosing FOSS for localization.

What is GNU/Linux?

GNU/Linux is a free Unix-type operating system originally created by Linus Torvalds and the GNU Project with the assistance of thousands of volunteer developers around the world. It is the most popular FOSS. Developed under the GNU General Public License, the source code for GNU/Linux is freely available to everyone. And one does not have to pay a licensing fee to download and use it. It is robust and secure, and it has no "hidden" features, because the source code is publicly available. Unlike proprietary operating systems, thousands of developers across the globe can inspect the code, identify vulnerabilities, provide security patches, contribute improvements and therefore historically GNU/Linux systems cannot be easily compromised by attackers.

Why Localize FOSS?

It is an acknowledged fact that the near-monopoly of English language software, controlled by English-speaking companies, does not serve the long-term needs of any country. Microsoft and a few other large US corporations dominate the international software market, earning large profits and wielding enormous power. For those with limited funds, the burden of paying for proprietary software means less money for other programmes of vital importance, as well as giving up linguistic freedom.

While proprietary software is often of the highest quality, policy-makers worldwide know that it carries a high risk of dependence on commercial corporations. If a corporation decides to no longer support software in another language, only those who are fluent in English would be able to operate computers effectively. And when local ICT professionals become proficient in both computers and English, they are quickly lured away from home, leading to a "brain-drain" that can damage a developing country for generations to come.

Key Advantages of FOSS Localization

- Reduced reliance on imports.
- No need for local users to learn English first.
- Local programmers gain expertise and experience.
- Local control over software appearance and functionality.
- New local technical standards and educational opportunities.
- Establishment of a local software industry. It is difficult for foreigners to do localization as they do not normally have an intuitive feel for the local language and therefore the language is compromised in most cases.
- National policy on local content would not be dependant on the availability of proprietary software or hardware.
- Localization of applications can be prioritized according to the national needs.

- Languages that are nationally important but financially unfeasible can be used.

Disadvantages of Proprietary Software

- Expensive to license and maintain.
- Dominated by the English language.
- Controlled by foreign corporations.
- Dependent on proprietary or closed standards.
- Has little or no local support.
- The high cost of the software leads to illegal copying of the software.
- The local software industry is not developed.
- Software cannot be localized or modified.

FOSS is generally much more secure than proprietary or closed source software. In the words of Peruvian Congressman Villanueva, "To guarantee national security, the State must be able to rely on systems without elements controlled from a distance. Systems with open source code allow the State and citizens to inspect the code themselves and check for back doors and spyware."[2]

Certainly, in an uncertain and sometimes dangerous world, few governments can afford to risk their information infrastructure security by relying on the goodwill of a secretive organization (such as a commercial software company) that they cannot control.

[2] 'The OSS advocacy report: Peruvian Congressman Responds to Microsoft', March 10, 2002; available from *www.studioforrecording.org/mt/Pubdomain_Bread/archivist/000013.html*

LOCALIZATION EFFORTS IN THE ASIA-PACIFIC

A Survey

In the winter of 2003, a survey was done on FOSS localization efforts in several Asian countries. Progress in localization varies considerably from place to place. Governmental sponsorship of the project appears to be the single most important factor in achieving rapid success in localization. Without exception, the developers surveyed cited "Freedom to Develop" as their primary reason for choosing to localize FOSS instead of using proprietary software.

The CJK Initiative

China, Japan and Korea are officially cooperating in FOSS localization. The well-funded and extensively promoted "CJK" programmes encouraging use of localized FOSS are at par with that of the wealthiest nations.

With educational and technological infrastructure and a large pool of skilled technology and language specialists, these three countries have the potential to dominate East Asian FOSS development in the next decade.

As donors and advisors, all three countries have a track record of supporting FOSS initiatives worldwide. Others should emulate the technical and organizational competence of the CJK FOSS and localization initiative, which is being led by the the Chinese Software Industry Association (CSIA), Japanese IT Services Industry Association (JISA), and the Federation of Korean Information Industries (KFII).

This primer is too short to fully detail the impressive work of the CJK initiative. However, a number of references to CJK as well as contact information for the key leaders of the movement are provided.[3]

Indian Languages

There are already multiple localized Indic language versions of GNU/Linux, Mozilla and OpenOffice.org available online. The depth of technical experience of Indian programmers in localization should enable them to provide assistance to other Asian countries in their localization efforts.

India has world-class IT infrastructure and skills. It also has a multiplicity of languages. Currently, the Indian Ministry of Information Technology is localizing to Hindi, Marathi, Kannada, Malayalam, Tamil, Telugu and Sanskrit.

At least two independent groups as well as the Ministry of Information Technology are currently localizing to Tamil.

An initiative to localize GNU/Linux to around 10 Indian languages, called Indix, is being spearheaded by the Centre for Development of Advanced Computing (CDAC), a scientific society of the Ministry of Communications and Information Technology, Mumbai. There is also a Hindi GNU/Linux and a Bangla version in progress.

English is widely used in India. Therefore, the underlying programming languages are easily understood, making the work much easier. For English language FOSS development, South Asia can realistically compete with Europe and the United States today.

[3]See *www.linuxinsider.com/story/32421.html*, *www.economist.com/business/displayStory.cfm?story_id=2054746*, *encyclopedia.thefreedictionary.com/CJK*, *www.chinadaily.com.cn/english/doc/2004-05/11/content_329529.htm* and *www.firstmonday.dk/issues/issue8_10/jesiek/*.

Thai

Localization of FOSS in the Thai language was started in 1993 by Thai students in Japan in an effort to use Thai in Unix environments. Later, a similar movement commenced in Thailand. These two movements merged and Linux TLE and the Thai Linux Working Group were launched in 1999. However, the movement was limited to a small community until mid-2001 when the Thai version of OpenOffice.org, dubbed Pladao and sponsored by Sun Microsystems (Thailand), was announced to the public. Documentation is being carried out by private publishers who, in exchange, will retain the copyright on their work for later sale to the public in the form of books and manuals.

Only 10 individuals are actively involved in the project at this time, with development resources provided by a combination of private companies and government.

Thai developers are using GNU Compiler and Tools for modifying GNU/Linux, Mozilla's "Bugzilla" for defect tracking, and CVS for version and source control. In addition, OpenOffice.org, a productivity suite, is being localized for Thai users.

The primary difficulties the Thais are experiencing include lack of feedback from users, lack of documentation, and conflicts with other Thai localization projects because of a lack of coordination and standards.

The Thais intend to roll out the software for both the private sector and public institutions, but are still in the planning stage at this time. The team has not yet seen a high-profile effort to promote the use of FOSS in Thailand. This may change in the near future, depending on government and private funding. In comparison to the CJK efforts, Thailand has fewer developers, less money, and less outside interest due to the size of its market. Other Thai localization projects are discussed on pages 7-10.

Vietnamese

FOSS localization to the Vietnamese language began in 1998, primarily as an individual effort by an enthusiast. Today, there are between 20-50 people working on the project. They are volunteers, providing their resources and time for free, and all are using FOSS tools for the work. But even with the number of individuals currently involved, there are not enough people to perform all of the necessary work.

Vietnamese localizations of GNU/Linux, Gnome, KDE, XFCE, Fluxbox, OpenOffice.org, Mozilla, XMMS, GnuCash, Gaim and Knoppix are in progress.

Plans for introducing the software to the Vietnamese people include the publication of books, formation of user groups in major cities, and provision of the software to private and public institutions. Already, there has been some effort to promote the use of FOSS in secondary schools, universities and government offices. Recently the government adopted a FOSS policy.

Malay

The Malaysian Institute of Microelectronic Systems (MIMOS) has launched a Web site to document projects in FOSS within Asia with links to key resources. MIMOS's goal is to localize key applications into the national language, Bahasa Melayu. MIMOS has already developed a local language GNU/Linux GUI and open source applications for the government. Two other localization projects are GNOME, led by Hasbullah Bin Pit, and OpenOffice.org, spearheaded by MIMOS.

Khmer

The Khmer Software Initiative has an ambitious though under-funded plan to localize FOSS for Khmer speakers in Cambodia and elsewhere. In addition to a localized version of the GNU/Linux user interface and some applications, they intend to create a development library, complete documentation both online and in print, and specialized training materials for developers and end users. It is not known how many individuals are working on the project currently.

As elsewhere, the Khmer team envisions providing the software to both private and government users. A publicity campaign is planned and the developers expect significant support from the international development community to complete much of the work.

The quality of the planning for Khmer, a crucial first step, is very high. As with other developing countries, funding and personnel to complete the work are the main hurdles. For more information on this area, please read Khmer Case Studies on pages 12-15.

Case Studies

Thai Localization

Fortunately for localizers, Thai language support has been stabilized by means of standardization achieved in the preceding years. The only limitation is the readiness of the internationalization frameworks within Gnome, KDE, Mozilla and Open Office.

FOSS communities of Thailand contribute a lot to both software development and user support. Governmental organizations also play important roles in FOSS reaching the masses.

Thai Language

The official language of Thailand is Thai. Spoken by almost the entire population, with several dialects in different regions, Thai is a tonal, uninflected and predominantly monosyllabic language. Most polysyllabic words in the vocabulary have been borrowed, mainly from Khmer, Pali and Sanskrit.

Thai belongs to the Tai language family, which includes languages spoken in Assam, northern Myanmar, Thailand, Lao PDR, northern Viet Nam, and the Chinese provinces of Yunnan, Guizhou and Guanxi.[4]

Thai script belongs to the Bhrami family. The oldest evidence of Thai script dates back over 700 years to the Sukhothai Age. Over the years, Thai script has gradually changed. Contemporary Thai script is composed of 44 consonants, 21 vowel symbols (32 sounds when combined), four tone marks, two diacritic marks and 10 decimal digits. Tone marks, diacritic marks and some vowel signs are stacked over or below the base consonants. The stacking is a shared characteristic among many South-East Asian scripts, especially those that are derived from Bhrami like Lao, Khmer and Myanmar languages. However, there are no complex precombined conjuncts in Thai, unlike most Indic scripts (Devanagari, for example). Only stacking is required. Lao is the script closest to Thai.

Standardization

Standardization is the key to the success of Thai language support in computers. It allows interoperability and resolves many localization issues. Important standards include character set, keyboard layout and input/output method specifications.

The standardization of IT in Thailand has been recognized since 1984,[5] when there were many efforts to use the Thai language in computers. More than 26 sets of *code pages* were defined by different vendors resulting in incompatibility. As a solution they were all unified as TIS 620-2529/1986 as the national standard by the Thai Industrial Standard Institute (TISI). A prominent legacy was the code table defined by Kasetsart University (KU) in a successful R&D effort to enable the Thai system in MS-DOS. It was the most widely adopted standard. Therefore, computer programs were obliged to support both encodings until TIS-620 became more popular and KU became obsolete.

Therefore, when Microsoft released Windows into the Thai market, TIS-620 was the only encoding adopted. The same was true for Macintosh. Thus, the character encoding issue was firmly settled.

In 1990, TIS-620 was amended to conform to ISO standards, but the code table was left completely unchanged. This new version was called TIS 620-2533/1990. The amendment enabled TISI to actively join many international standardization activities. For example, it submitted the character set to the European Computing Manufacturers' Association (ECMA) for registration in the ISO 2375 repertoire, and was assigned as ISO-IR-166 so that it could be used with ISO/IEC 2022 mixed code page encoding. An example of such implementation is GNU Emacs. Around 1996, a TISI technical committee drafted a proposal for the Latin/Thai part (part 11) of ISO/IEC 8859, based on TIS-620. However, it was suspended due to the prohibition of combining characters. It was reactivated in 1999, and endorsed as an international standard in 2000.

[4] Thai Language Audio Resource Center, 'Some Historical Background of Thai Language'; available from *thaiarc.tu.ac.th/thai/thai.htm.*

[5] Koanantakool, T. and the Thai API Consortium, *Computer and Thai Language*, National Electronics and Computer Technology Center, 1987, in Thai.

The TIS-620 code table was pushed for inclusion in the Unicode table. Although the influence of the ISCII encoding scheme (which forced all vowels, including the leading vowels, to always be encoded after consonants) had made the Unicode consortium force Thai to change its encoding scheme, TISI defended the TIS-620 practice, as Thai script did not need such complications. Although this made Thai (and Lao) different from other Indic scripts, it saved Thai (and possibly Lao) implementations from the big hindrance of lacking supporting technology for ISCII practice at that time, as well as from the burdens of migration from the well-settled and widely-adopted practice. So all Thai language-processing codes for TIS-620 and Unicode, apart from the necessary code conversion, are fully compatible.

In addition to the character set, the Thai keyboard layout was standardized as TIS 820-2531 in 1988, and later amended by adding keys for special symbols and published again as TIS 820-2538 in 1995. Another keyboard variant designed after research on character frequencies, called Pattajoti, is also available. But it is not as popular and is not a national standard.

Thai input/output methods were also standardized through the efforts of the Thai API Consortium (TAPIC), which was formed by a group of vendors and academies and sponsored by the National Electronics and Computer Technology Center (NECTEC). The specification, called WTT 2.0 (from its Thai abbreviation 'Wor Tor Tor', which stands for "Wing Took Tee" or "runs everywhere"), was published in 1991. Its contents were composed of three parts, describing character set and encoding scheme, input/output methods, and printer identification numbers.

Although not endorsed by TISI as a national standard, WTT 2.0 was adopted by virtually all the major vendors who participated in the draft, including DEC, Sun, Microsoft, MacIntosh and IBM. WTT 2.0 had enjoyed being the *de facto* national standard for seven years, until it was dubbed TIS 1566-2541 by TISI in 1998.

There are other activities[6] with international standards bodies that promote understanding of Thai language requirements among vendors. For example, in 1998 the "tis-620" MIME character set was registered with the Internet Assigned Number Authority (IANA) for information interchange on the Internet.[7] Another example is an annex of ISO/IEC 14651, International String Ordering, describing how the predefined sorting order for Unicode can be tailored to match the requirements of Thai string ordering.

With these established standards, specifications for Thai implementations are clear and interoperability is guaranteed. The standards have played an important role in several developments in the Thai computer industry, including FOSS localization.

Localization

Thai localization of FOSS was started in 1993 by Thai students in Japan with the *ThaiTeX* project initiated by Manop Wongsaisuwan as a first effort to use Thai in the versatile typesetting program.[8] Subsequently, the project was maintained by the Thai Linux Working Group (TLWG)[9] which was formed in 1999.

Apart from Thai LaTeX, other surrounding UNIX environments have been modified by the same group of people to support Thai. Their work may be summarized as follows:

- *Manop Wongsaisuwan:* ThaiTeX (ttex), X bit-map fonts.
- *Thai Project (by Vuthichai Ampornaramveth):*[10] cttex (C version of ttex), xiterm+thai (Thai X terminal), likit (Thai text editor for X).
- *ZzzThai project (led by Poonlap Veerathanabutr):*[11] thailatex-component, X bit-map fonts, Thai support in xfig, Thai-HOWTO and Thai RPMs.

At the same time, other Thai support projects were developed by researchers of the NECTEC in Thailand, including:

- *Virach Sornlertlamvanich:* Thai support in Omega (Unicode-based TeX kernel), Thai in GNU. Emacs, machine translation, and many other NLP projects.

[6] Karoonboonyanan, T. and Koanantakool T., *Standardization Activities and Open Source Movements in Thailand,* Country Report, MLIT-4, Myanmar; also available at *www.nectec.or.th/it-standards/mlit99/mlit99-country.html.*

[7] Tantsetthi, T., 'Campaign for Internet-Standard-Conforming Thai Usage'; available from *software.thai.net/tis-620.*

[8] Wongsaisuwan, M., 'Introduction to ThaiTeX'; available from *thaigate.nii.ac.jp/files/thaitex.pdf.*

[9] Thai Linux Working Group, 'Thai LaTeX'; available from *inux.thai.net/plone/TLWG/thailatex.*

[10] Ampornaramveth, V., 'NACSIS R&D Thai Project Page'; available from *thaigate.nii.ac.jp.*

[11] eucthai, 'ZzzThai Project'; available from *www.fedu.uec.ac.jp/ZzzThai/.*

- *Surapant Meknavin:* thailatex (babel-based), Thai search engine.
- *Phaisarn Charoenpornsawat:* swath (Thai word break utility).
- *Theppitak Karoonboonyanan:* Thai string collation , Thai Locale.[12]
- *The National Fonts Project:* Standardized font design specification, three public domain vector fonts (Kinnari, Garuda and Norasi).

Another project worth mentioning is Linux SIS (School Internet Server),[13] which was initiated by NECTEC for use in the SchoolNet project.[14] Although it is a server-centric distribution, it was during this project that another main task force for Thai FOSS localization was constituted. Through a mailing list for supporting its users, the volunteers agreed that another distribution for desktops was needed, and was feasible to develop, as almost all of the specialists mentioned above were there. This task was undertaken by the *Thai Linux Working Group*. A Web site (*linux.thai.net*) was created for general user support. A new GNU/Linux distribution called Linux TLE (Thai Language Extension) was created to collect, as comprehensively as possible, the existing works of Thai developers and package them for users.

Apart from being a tool for boosting the use of FOSS by Thai users, Linux TLE also provided a platform for development, and a test cycle where users could participate through bug reports. The ultimate goal was to improve Thai support for FOSS from the source. Therefore, getting patches checked-in to upstream projects was the final success indicator.

So far, lots of source code from TLWG and Linux TLE have been incorporated in upstream projects, including:

- Thai locale definition in GNU C library.
- Thai keyboard maps in XFree86.
- Thai XIM in XFree86.
- Thai fonts in XFree86.
- Thai Pango modules.
- Thai string ordering in MySQL.
- GTK+, GLib, Qt, KDE, Mozilla, Xpdf, etc.

In 2000, Linux TLE was handed over to NECTEC for maintenance. Three versions (3.0, 4.0 and 4.1) were released and gained a lot of recognition from users in all parts of the country. A dedicated Web site was created for it at *opentle.org* in 2003. TLWG continued to build its user and developer communities.

Some of the TLWG projects that are hosted and maintained by the community are:

- *libthai :*[15] a library of Thai basic support routines, including character sets conversion, input/output methods, word break, string collation, etc. Some plug-ins for Pango GTK+ IM are also provided.

- *thaifonts-scalable :*[16] a collection of scalable fonts available to the public, plus some fonts developed in-house. All fonts are maintained and improved based on standard technical specifications.

- *thailatex :*[17] Babel-based Thai support for LaTeX document preparation, based on Surapant Meknavin's work.

Other significant development efforts to boost the Thai environment in FOSS are Pladao[18] and OfficeTLE.[19] Both projects aim to develop Thai support in OpenOffice.org. Pladao was initiated by Sun Microsystems (Thailand) and was subsidized by Algorithms Co. OfficeTLE was initiated and operated by NECTEC. Both

[12]Karoonboonyanan, T., 'Thai Sorting Algorithms'; available from *linux.thai.net/thep/tsort.html;* Karoonboonyanan, T., Raruenrom S. and Boonma P., 'Thai-English Bilingual Sorting'; available from *linux.thai.net/thep/blsort.html;* Karoonboonyanan, T., 'Thai Locale'; available from *linux.thai.net/thep/th-locale.*

[13]National Electronics and Computer Technology Center, 'Linux SIS: Linux School Internet Server'; available from *www.nectec.or.th/linux-sis.*

[14]National Electronics and Computer Technology Center, 'SchoolNet Thailand'; available from *www.school.net.th.*

[15]Thai Linux Working Group, 'LibThai Library'; available from *linux.thai.net/plone/TLWG/libthai/ and libthai.sourceforge.net.*

[16]Thai Linux Working Group, 'ThaiFonts-Scalable'; available from *linux.thai.net/plone/TLWG/thaifonts_scalable/.*

[17]Thai Linux Working Group, 'Thai LaTeX'; available from *linux.thai.net/plone/TLWG/thailatex/.*

[18]See *www.pladao.org.*

[19]See *opentle.org/office-tle.*

of these projects are working in parallel, emphasizing different aspects. Pladao is feature-rich, while OfficeTLE emphasizes Thai language processing quality. Many hope that they will merge, possibly through upstream projects.

Obstacles

These are the obstacles to the localization of FOSS in Thailand:

- *Too few developers.* With the growth of the user base, the lack of FOSS developers to serve the growing requirements and expectations is a big problem. The premature growth of FOSS adoption in Thailand has unbalanced the community, both in terms of the ratio between users and developers, and in terms of unrealistic expectations with regard to price and freedom. Worse, the community existing developers have been fragmented because they are often employed by competing businesses. Proprietary competition has reduced the traditional cooperation responsible for the progress of FOSS in Thailand.

- *Misconception.* If one wishes a movement to be initiated by the government, it is necessary for the government to have a good understanding of all of the concerns. The government must realize the benefits of FOSS as a means for developing technologies, as well as for improving the technical skills of the developers. Otherwise, it just becomes exploitative. According to some, even though the Thai government has popularized FOSS through the recent campaign for affordable PCs, they have also damaged public opinion of FOSS by their failure to provide proper support. Premature promotion of FOSS to uninitiated users when it is not ready would only create a bad impression. On the other hand, claiming that Microsoft's decision to lower the price of proprietary software is actually a success for FOSS, can be seen as exploiting FOSS as a negotiation tool. Moreover, "localization" tends to be perceived by the government as an activity that is about development of local GNU/Linux distributions, which is not true. Thus, many government policies simply miss the point and sometimes make things worse.

Lao Localization

"Laonux" is the name of the Lao language version of GNU/Linux. This section deals with Laonux implementation. An overview of its success and obstacles to its development is also provided.

Traditionally, the Lao language and its literature have been written in two scripts, Lao and Tham. The Tham script is derived from the Lanna script (of present-day Chiang Mai and northern Thailand), which in turn originates from ancient Mon.[20] The Lao language belongs to the Tai language family, which includes languages spoken in Assam, northern Myanmar, Thailand, Lao PDR, northern Viet Nam, and the Chinese provinces of Yunnan, Guizhou, and Guanxi.[21] Lao script is believed to have originated from the Khmer writing system. It originated in the Grantha script, which was the southern form of the ancient Indian Brahmi writing system.[22]

Lao script shares characteristics with other South-East Asian writing systems. It follows complex rules of layout involving consonants, vowels, special symbols, conjuncts and ligatures. Spaces are not used to separate words, and vowels appear before and after, under and over consonants.

Obstacles and Successes

In localizing FOSS for the Lao script, the following tasks must be completed:

- Identifying the technical obstacles to be overcome.
- Creating the world's first English/Lao technical dictionary.
- Finding and coordinating technical volunteers.
- Establishing *de facto* technical standards.
- Performing and testing the work.
- User training and education.
- Identifying the sources of funding.

The work of investigating how GNU/Linux can be localized for Lao began in the summer of 1999 and

[20]See *www.lan-xang.com/literature/lit_3.html.*

[21]Thai Language Audio Resource Center, 'Some Historical Background of Thai Language'; available from *thaiarc.tu.ac.th/thai/thai.htm*.

[22]See *seasrc.th.net/font/alphabet.htm.*

proceeded slowly at first. Months of research and experimentation identified the technical difficulties and tools available to resolve them. Anousak Souphavanh, working from Rochester, New York, eventually formed a team of volunteers, teachers and students from the National University of Lao PDR to focus on translating KDE into Lao. The KDE desktop in Lao allows the end user to access the basic functionalities of a computer, including email, Web browser and office applications.

Until 2002, the work progressed slowly but steadily, aided largely by helpful and frequent advice from fellow FOSS enthusiasts worldwide. The community regularly provides invaluable information and assistance in tracking down documentation of technical issues, relating experiences in solving similar problems, or simply encouraging the volunteers in their efforts.

In 2002, the Jhai Foundation provided a small stipend to support the development of Laonux. As before, research had to be performed on the technical issues and tools, and more volunteers had to be found for the required translation and programming. The programming work was completed in 2003. However, the bulk of the translating remains undone. The major obstacle is the lack of an English/Lao technical dictionary. Without this, any translation will be inconsistent and confusing to users.

Upon completion of the dictionary, it is relatively simple to translate the remaining message strings and integrate them into the desktop environment. This work can be performed by either professional or student translators at a relatively low cost. But until the dictionary is completed, the work will continue at a snail's pace.

Funding for the dictionary and translators is now being sought from international development agencies. In addition, funding for professional documentation, training and user education is required.

Standardization

The lack of technical standards for the Lao script and its implementation in software remains a difficult challenge. The following standards must be completed and accepted officially:

- Character set.
- Keyboard layout.
- Unicode fonts.
- Input methods.
- Output methods.

To date this project has yielded some very useful *de facto* technical standards. Lao officials now recognize the vital importance of these standards and have adopted the goal as their own. These standards and the technical dictionary could dramatically speed up the ensuing. Until the majority of Lao developers agree to follow standards, further development will be hampered.

Localization

Team members have resolved the following issues. These solutions follow ISO standards and should be adopted by future localization efforts to avoid duplication of effort and incompatibility between systems.

First, Unicode fonts were used instead of the existing fonts that override glyphs of English letters as per keyboard layout because these fonts are not standards compliant. Unicode is now the worldwide standard for FOSS developers. Also, development tools such as Kbabel and KDE require Unicode compatibility.

Laonux is a KDE localization, which is why the rendering of fonts is handled by the Qt libraries (as opposed to Pango or X Window). The Qt library file "Qt-qfont_X11.cpp" was modified slightly to point to the appropriate range for Lao. Initially Lao fonts were not rendered properly while stacking combined characters (i.e., consonant and a vowel). With the help of Theppitak Karoonboonyanan, patches were submitted to fix the rendering. Resolving these issues was a major breakthrough for Laonux development.

Inputs are handled by XKB, the keyboard map for X. It is used in converting key-strokes to key symbols defined in "X11/keysymdefs.h", with language switching capability. XKB Lao keys were created with the appropriate Lao Unicode range. Finally, the GNU C library definition for Lao was created. This is mainly UTF-8 locale settings for date/time and related issues.

Future Plans

- Continue work on Laonux.
- Begin work on localizing OpenOffice.org.
- Continue translation work for PHP Web portal tools.

- Begin work on localizing Mozilla and other Web tools.
- Train users and technical staff for localization projects.

The first priority is to create the English/Lao technical dictionary. Whether this will be funded by governmental sources, grants or international aid is unclear. Without this dictionary, translating and localizing the software cannot be completed. Another priority is to continue cooperating with other regional software localization efforts, especially where they have already addressed similar issues.

Additional modifications and updates to ISO and Lao government IT standards are required for the long term. These include Input/Output methods, keyboard layout, collation, locales, and additional Lao OpenType Unicode fonts standards.

Cooperation with universities for localization is ongoing. With most of the technical issues resolved, localization is now primarily a language issue. To avoid unnecessary anglicisms, professional linguists are needed. If technical professionals are involved in this process, they are likely to impose what they already know, which is English. It is far better for the future to adopt native language terms wherever practical. However, without funding, it is likely that a hodge-podge of anglicisms will prevail.

On the technical level, support for Lao script in IBM's ICU library is needed. ICU is the script support base for OpenOffice.org and for Java (released by IBM and Sun Microsystems). It is a complete library, including script rendering, layout, collation (sorting of words), line breaking, spell-checking, etc.

Some difficult and important technical work has been done. However, full integration in ICU will force us to define all of these issues very clearly, and this will pave the way for the developers to create Lao applications in Java and C++. Without this, professional Lao software development can be difficult.

Khmer Localization[23]

Khmer Language

As in the case of Thai and Lao, Khmer script originates from the Grantha script, the south Indian form of the ancient Indian Brahmi writing system.

Khmer script follows complex rules of layout in which consonants may take two different forms (e.g., the small form is placed on a lower line if it immediately follows another consonant). Space is used not to separate words but to indicate a pause in reading (very much like a comma in English). Vowels pronounced after a consonant may appear before, after, above, below; before and after (formed by two glyphs); before and above; below and above; or under and after the consonant.

At present, the definition of the language is so poor that even the number of vowels in the language is not clear. The number of vowels in the official reference (the only available dictionary) is different from the number of vowels taught in schools. The reference dictionary is sorted phonetically, making a systematic collation algorithm that will follow the same order impossible. Words starting with the same consonant may be ordered under different listings depending on how that consonant is pronounced in that word. As in the Lao localization project, an English/Khmer technical dictionary is not available, and the lack of it severely hampers the efforts to translate software into the local language.

Obstacles and Successes

When the KhmerOS project was first being considered, the technical situation was as follows:

- Khmer had already been included in Unicode. Fixed in 1996 by a team of people who had no contact with the Cambodian government, the definition in Unicode was later disputed, but to no avail. The Unicode Consortium refused to change anything, including the addition of necessary Khmer vowels, on the basis that these could be formed by combining other existing Khmer characters. The Consortium only permitted adding comments to the existing standard. The standard is now considered fixed by the Khmer government (in its 4.0).
- Microsoft had published OpenType specifications for Khmer, and included the language in its Uniscribe complex text layout engine. MS Publisher worked very well in Khmer, but MS still did not handle either line-breaking or sorting. MS Word crashed quite often while using Khmer.
- Some OpenType Khmer fonts already existed, though none in the public domain.

[23] This case study has been contributed by Javier Sola

- No FOSS programs were implemented in Khmer in the GNU/Linux environment, but some FOSS (such as Mozilla) worked well in Khmer under Windows, using the Microsoft Uniscribe engine.
- Some people had been considering FOSS in Khmer, but the idea had not gone beyond mailing list discussions.
- There has been an amazing proliferation of legacy (non-Unicode) fonts. Up to 26 different font encodings had been defined. They worked well enough under MS Word by modifying "normal.dot".

It is also important to understand the social situation, as follows:

- The lack of computers in the Khmer language increases the digital divide. Only people who speak English have access to jobs that require the use of computers.
- Because computer interfaces are in English, the words used for computer-related work are also in English, introducing many "anglicisms "that could have been easily avoided if the software had been translated.
- As people have to memorize new English words (in the menus), training for computer use takes a long time and it is soon forgotten if it is not used.
- There is no English/Khmer technical dictionary.
- The development of databases for governmental purposes is not possible, as no database management systems that handle either legacy or Unicode Khmer encodings have been developed.
- In order to join the World Trade Organization, Cambodia has passed an intellectual property law that, when in force will, in theory, compel people to pay for proprietary software. FOSS allows developing countries to comply with the requirements of WTO without having to spend large amounts of money.

Spanish computer scientist Javier Solá, who lives in Cambodia and has more experience with computers and strategy than with FOSS, has decided to see what can be done in this situation. He has started writing an ambitious project with the following deliverables:

- A full computer operating system, office suite and entertainment applications needed by an average computer user – entirely in the Khmer language. A user will see only the Khmer language script on his/her screen. The system will include full documentation in Khmer, in electronic and paper formats.
- A "development library" (a set of programs) to be used by software developers to include support for the Khmer language in their applications; documentation of the development library.
- A set of up to 50 computer fonts (Unicode OpenType fonts) to be used in the application menus, for word-processing or for computer design.
- A keyboard layout, supporting drivers and 5,000 physical keyboards that support the above-mentioned Unicode fonts.
- 5,000 copies of an installation disk that would be easy to use and would include all of the above software and documentation; with 1,000 of them accompanied by a full printed documentation.
- Training materials addressed to end-users, including the use of the system and the applications, and a training course for typing with the new keyboard.
- Computer end-user trainers trained to teach the new system using the above-mentioned materials: University professors, students and software development company personnel trained in advanced GNU/Linux and FOSS and application development using the Khmer script support tools provided by the project; computer vendor personnel trained in the installation of the system.
- FOSS expertise centres in universities, including trained professors, students and computers with Internet connectivity.
- Software development companies empowered to develop applications based on FOSS that require support for the Khmer language script.
- Government personnel trained to use the system.
- 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 directly or through computer and software vendors.

The project not only considers the final goal but also the order in which the programs will be translated. First Mozilla, an email client, and then OpenOffice.org, an office suite, will be released under MS Windows. Finally these will be included in a fully Khmer GNU/Linux release when the user interface is translated.

After considering and later rejecting (for lack of real industry interest) the creation of an Industry Consortium, the project soon found a home in the local Open Forum of Cambodia, an NGO with a long history of supporting technologies for social purposes in Cambodia.

Sharing the idea of the project with anybody who would listen, setting up a Web site (*www.khmeros.info*), and initiating contact with various people interested in Khmer and computers, brought in the first volunteers.

Typographer Danh Hong put into the public domain one of his Khmer OpenType fonts. This was an important preliminary step to get the FOSS community interested in this language.

Lin Chear, a Canadian engineer of Khmer origin, started looking at Pango and in a few days created the necessary programs to support Khmer in Gnome and its family of products. Khmer will be supported by version 1.6 of Pango. KDE followed soon after.

With the help of Lin Chear, the European maintainer of Qt, developed the routines needed for supporting Khmer in KDE and programs that use Qt for language support. Lin Chear applied a Pango patch to Mozilla and got a version of Mozilla working in Khmer on GNU/Linux.

Back in Phnom Penh, the KhmerOS project established itself in a small office at the Open Forum facilities. Using USD1,500 from the first small donations, they hired two computer scientists and with a couple of old donated computers, started creating a glossary of Khmer computer terms as preliminary work before starting the translation. A donation from a Hong Kong businessman would later permit the purchase of new computers.

Meanwhile, work and discussion regarding keyboards and collation algorithms are ongoing. Implementations in the Thai language show that dictionary based line-breaking can be done in languages that do not separate their words. Work on conversion of texts in legacy fonts to Unicode encoding has also started.

"Evangelization" for Unicode is a necessary part of the project's work. Large amounts of money are donated to Cambodia or used by NGOs for computer-related projects, such as creating a database of all Cambodian laws. If these projects are not done in Unicode, the databases will be useless in a few years.

Future plans include increasing the project team to five translators with at least one of them a professional translator, and releasing email, browsing, word processing and spreadsheet programs in Khmer (Thunderbird and Firefox from Mozilla and OpenOffice.org Writer and Calc).

These programs will all be released on MS Windows (2000 and XP), and will have full documentation in Khmer, as well as basic training modules in Khmer to be used by computer training professionals. Training materials are important in order to fix the language and terminology used by teachers, assuring standardization and avoiding use of too many ''anglicisms".

The translation of Gnome or KDE (or both) began in 2004, as well as a series of minor applications that will allow the release of a complete GNU/Linux-based system in Khmer in the second half of 2005.

Funding is still a major concern. Project speed is adjusted to suit funding, which comes either through grants or through contracts with corporations or other NGOs that need translated programs or services related to Khmer Unicode that can be provided by the project itself.

On the technical level, support is still required for Khmer script in IBM's ICU library. Full integration in ICU will compel the project to define all such issues clearly, as well as open the way for developing Khmer applications in Java.

Another concern is being able to use low-priced computers. In the Microsoft environment, Khmer Unicode will work only in Windows 2000 and XP, which require modern computers with a fair amount of memory and large hard disks. The way to low-priced (i.e., secondhand) Khmer computers will definitely have to be either through GNU/Linux or by getting Khmer script support in earlier versions of Windows such as Windows 98.

The project's distribution strategy is to try to have the software pre-installed by computer vendors and distributed by "two-dollars-a-CD" software vendors. This is to allow easy copying and wide distribution, thereby saving the project the cost of making copies.

The government, under an IDRC grant, is pushing for the localization of Windows and MS Office. They have neither plans nor resources to get involved in FOSS localization, but are said to be interested in coordinating efforts so as not to duplicate work and to ensure that the Khmer language (collation, etc.) is implemented consistently in both platforms.

The project aims not to be sectarian about FOSS. It is better to have people using email in Khmer on Windows in the short run than to try to make them change to a completely new system abruptly. There are still two or three years before the intellectual property law is expected to be strongly enforced. This period can be used as transition time, assuring a gradual success rate, rather than trying to force an overnight change and thereby failing.

Status of Localization Projects

A survey[24] of the status of localization projects was conducted by the PAN Localization Project[25] during the training on "Fundamentals of Local Language Computing" in Lahore, Pakistan in January 2004. Participants from 13 Asian countries were asked to provide information about the status of localization tasks in their countries. The survey revealed the status of standardization, localization and national policy for local language computing of the participants' countries. Please note that the informal survey technique employed gives only a rough picture of the actual situation.

Table 1 summarizes the survey on standardization for character set, keyboard layout, key-pad layout (e.g., for mobile phones), collation sequence, terminology translation and locale definition.

Table 1 PAN Survey: Localization Standards

Country	Language	Char Set	KBD	Keypad	Coll. Seq.	Interface Term.	Locale
Myanmar	Burmese	✓				*	*
Cambodia	Khmer	✓	✓				
Mongolia	Mongolian	✓	✓	?	*	*	✓
Lao PDR	Lao	✓	✓				
Nepal	Nepali	*	*		*	*	*
Sri Lanka	Sinhalese	✓	✓	*	*		*
Thailand	Thai	✓	✓	*	*	*	✓
Bhutan	Dzongkha	✓	✓		*		*
China	Tibetan	✓	✓		?		
Japan	Japanese	✓	✓	✓	✓	✓	✓
Bangladesh	Bangla	✓		?	*	*	✓
Afghanistan	Pashto	✓	✓		?		*
Iran	Farsi	✓	✓		*	*	*
Pakistan	Urdu	✓	*		*	*	*

*✓: complete; *: partially done; ?: don't know; blank: no work done*

From the table, character set and keyboard layout seems to be well defined for most countries, while other issues need more work.

Table 2 summarizes the current status of localization of applications on the GNU/Linux platform, namely, keyboard input, fonts, sorting and find/replace utilities, natural language processing, spell checker, and thesaurus. It also shows whether any GNU/Linux distribution has been released in the country.

[24]Hussain, S. and Gul S., *PAN Localization Project: A regional initiative to develop local language computing capacity in Asia*, 2004.

[25]See *www.panl10n.net/.*

Table 2 PAN Survey: Basic Localized Applications on GNU/Linux

Country	Language	KBD Driver	Font	Coll.	Find / Replace	NLP	Spell Check	Thes.	Linux Distr.
Myanmar	Burmese		*						
Cambodia	Khmer								
Mongolia	Mongolian	✓	✓	*	*		*	?	✓
Lao PDR	Lao								
Nepal	Nepali	✓						*	*
Sri Lanka	Sinhalese	*	✓				*		*
Thailand	Thai	✓	*	✓	✓	*		*	✓
Bhutan	Dzongkha								
China	Tibetan	*			*		?		
Japan	Japanese	✓	✓	✓	✓	?	?	?	✓
Bangladesh	Bangla	✓	*	*				?	*
Afghanistan	Pashto		*		?	✓			
Iran	Farsi	✓	*	✓	*	✓	*	*	*
Pakistan	Urdu	*	*			✓			*

✓*: complete; *: partially done; ?: don't know; blank: no work done*

The survey shows that FOSS localization activities are taking place in many countries. Sharing expertise and resources among countries can boost progress in this area.

For message translation, a number of Asian language localization projects are underway. All of these projects are tracked in real time on the Internet. The following links provide the current status of some of these projects.

- *i18n.kde.org/stats/gui/HEAD/fullinfo.php*
- *www.mandrakelinux.com/l10n/status.php3*
- *l10n-status.gnome.org/gnome-2.6/index.html*

As of June 2004, Japanese and Korean translators have already completed most of their initial work in FOSS localization. China is close behind, followed by India and the South-East Asian nations. The situation is highly fluid, however, and the above sites should be checked for detailed and updated information on the current status.

RECOMMENDATIONS

Implementing Localized FOSS in Asia

Asia is the world's battleground for FOSS localization. The monopolistic proprietary companies have not yet established dominance in Asia. For reasons of national interest, many Asian governments have adopted policies that encourage software alternatives, primarily FOSS.

In some countries the localization to date has been the work of a few dedicated enthusiasts scattered around the globe. Few have been paid for their efforts, and the disorganized nature of their translations has unintentionally produced ambiguities.

For the most part, the volunteers are programmers, not linguists. They need help from translators, technical writers and testers. Since localization deals primarily with language rather than programming issues, non-technical staff should outnumber programmers five to one. Even before adopting a formal FOSS policy, direct support for technical dictionaries and standards for localization can begin. Local language specialists require only professional salaries and offices. Technical writers and testers can be trained within a few months.

Supporting international localization initiatives, where languages that share significant similarities (i.e., Thai, Lao and Khmer) share programming and technical resources, is very cost effective. The value created for society as a whole, with new dictionaries and a technical standard enabling programmers and translators to consistently localize any FOSS for a low price, is undeniable. Other Asian countries should follow the example of the CJK initiative.

Localization initiatives should have very clear objectives, and the resources required to meet those objectives. They require funding, professional management and technical expertise. In addition, thorough linguistic knowledge is critical to success.

These initiatives will result in the creation of local centres where the knowledge is dispersed to those who will perform the actual work. Such centres can be the product of governmental action or business partnerships, or operate as part of a university. Regardless of how the centres are founded, they should enjoy the full support of government policy.

It is time to professionalize the process of Asian software localization, especially for developing countries. A great opportunity can be lost if haphazard efforts lead to undependable results. Where public good cannot be achieved by individual effort, the government is expected to help. A professional group can request the help of volunteers, but central coordination and basic work ought to be done by a dedicated team of paid staff.

It could be that the yearly fees paid by a developing country for a single department's commercial software is enough to underwrite that country's participation in the FOSS movement.

Establish localization centres to be the focal point for FOSS developers to share information, develop skills, and build on existing accomplishments. Where different countries have linguistic commonalities, a regional localization centre could share the cost of development. Specialists who are familiar with source code, linguists and analysts could be available to assist a wide variety of projects and build a knowledge base to accelerate future development.

Sponsor the creation of technical dictionaries and standards so that consistency is retained in all FOSS projects. With standard terminology, computer users are less likely to encounter frustration, and with standard technological procedures and processes, FOSS code can remain comprehensible to all IT professionals. Adherence to such standards should be mandated in all software procurement policies implemented by the government.

Move fast. It is important to have things done correctly, but it is also important to do them quickly. Writing a good computer glossary for a low-technology language can take over a year, but a first glossary that will suffice for translating the first versions of the programs can be done very quickly (e.g., in three months). Future versions of the programs will use the final glossary, but first versions can be available within months. An official "portal" detailing the prescribed terminology and standards should be the first priority.

Encourage the distribution of FOSS operating systems, applications and platforms. With little cost, governments can distribute localized FOSS to schools, businesses and other organizations. This would jumpstart the rate of adoption of computers and software in general, and prevent the unnecessary illegal copying of proprietary software. Because FOSS often works with older machines, the total price of providing computing access to the masses would be lower than that for any other approach.

Provide FOSS training not only for computer professionals, but also in primary and secondary schools. In developing countries where educational budgets are stretched thin, the use of localized FOSS operating on low-cost computers is well suited for increasing educational opportunities in rural communities. The natural curiosity of the youth should quickly result in a new generation that knows how to use computers in their native language. Those students who show a special talent for using computers can be encouraged to learn programming through scholarships, contests and other age-appropriate activities.

Beyond establishing governmental purchasing policies that favour localized FOSS, governments have an important role in removing obstacles, providing funding and coordinating standards. Without governmental support, "anglicisms" and inconsistencies will severely hamper the continued localization of FOSS, and limit the possibilities for growth of an indigenous software industry.

Skills and Tools Required for Localization Projects

Localization often occurs when the country is already using computers in a foreign language. Computer scientists and trainers are used to an English or French computer vocabulary. Localization therefore requires creating training materials based on the language used in the glossary, so that trainers and new users will start using the local language.

As it is difficult to engage linguists, preparatory work can be done first, such as looking for different translation options for each term.

After this, the work is mainly that of translators, who follow glossary guidelines and rules. There should be professional translators and computer scientists in the same team to assure linguistic and technical correctness of the terms used.

Localization can increasingly be performed without too many technical resources, once the first layer of the work is done (fonts, language support, etc.). In the future, it will become easier, since almost all FOSS projects are adopting new tools and techniques to make it easier for non-experts to perform the work.

The skilled workers who can perform software localization are often already available, or can be trained locally or abroad. Regional software localization training and coordination centres could act as clearing-houses and colleges for individuals to improve their skills, and thereby produce new workers for the years ahead. Fortunately, only the programmers need to have specialized knowledge of FOSS. The other professionals can have previous experience with any type of software.

Office space that is sufficient and appropriate for the work at hand is a must for any project where work is not distributed ad hoc around the world. For a professional localization effort, and especially for multilingual regional localization centres, a commercial space is best. This includes stable low-cost broadband connections to the Internet, LAN and development servers, sufficient client computers for each employee and three or four terminals for each tester.

Active participation and cooperation from universities, especially linguists and translators of English, should be solicited. Publishing rights for scholars who make significant contributions to technical dictionaries and standards should be granted, as well as public recognition for student volunteers.

Typically, the following people need to be trained, organized and provided with the tools to succeed:

- Project managers – technical and translation.

- Analysts and linguists.
- FOSS programmers.
- Translators and technical writers.
- Testers.
- Trainers.

Project management for localization should be split into two jobs: (i) Technical Managers direct the actual editing of code to ensure proper language support; and (ii) Translation Managers coordinate the creative efforts of linguists, technical writers and trainers.

Analysts and linguists work together with project managers, sociologists and programme sponsors to identify the technical challenges to be overcome and the cultural-linguistic requirements to be met. Their work results in requirement specifications and a project description that the project manager uses to guide the project to completion. This roadmap guides the programmers in their work, and provides the benchmark against which the software will be tested. The analysts are also responsible for gathering, organizing and disseminating the technical standards and specifications required by the programmers to perform their work.

Since both the operating system user interface and various application user interfaces should be localized, often several different types of programmers will be required. Enthusiasts can perform this work remotely with others worldwide, but only if the problem has been thoroughly documented by the analysts. Wherever possible, local programming staff should be used for this stage of the work. The lessons they learn, and write down for later reference, can be spread to others who are performing localization. They may also create the technical standards for that language if none exists. Compliance with such FOSS standards as "G11N, I18N, L10N" (please see Glossary) and others, will ensure that work proceeds quickly with the confidence that successive developers can continue to update and improve the software.

Translators and technical writers perform the lion's share of the work. All the error messages, buttons, menus, commands, help files and user guides must be translated. In consultation with linguists for consistency and accuracy, translators and technical writers compile technical dictionaries, often coining new technical words and phrases that enable future developers to communicate more effectively with their colleagues. Just as the technical standards for localization are vital to programmers, the technical dictionary used by the writers and translators is vital to the project's success.

Testers use the requirement specifications to check the complete work of both the programmers and technical writers. Their painstaking work identifies errors and inconsistencies to be corrected, and rechecked, before release to the users of the software. Additional apprentice testers, especially those who speak no English and who are computer novices, can provide excellent feedback for programmers and translators.

Trainers introduce the localized software to the users. Often, local teachers who have been taught how to use the system give seminars, answer questions and mentor computer enthusiasts. Local businesses and governments may also hire trainers to educate their workforce. It is important to ensure that these software trainers are locally recruited and speak the native language, rather than being English speakers imported at great expense.

For both proprietary and free software, training on how the software works is essential. To teach local users how to operate the software, one needs:

- Training equipment and materials.
- Classrooms.
- Instructors.

Most often, developers of the software 'train the trainer', who then instructs novices to make them advanced users. Training can be further divided between user training, system administrator training and developer training. Except for user training, which should be widespread, most specialized FOSS training takes place in educational institutions. Countries that have advanced quickly in FOSS localization have all devoted considerable resources to training and education. Without actual adoption of the software by a large segment of the population, the work of localization is an exercise in futility.

Tools and equipment for FOSS development and localization are less expensive than those required for proprietary localization. Version control, project management, documentation change management, and development tool kits for programmers are all available either free of charge or at a low cost. To work on FOSS, experience shows that it is best to use free/open source tools.

All other equipment, including most development computers, should be up to date, in a secure environment. A separate budget should be set aside for libraries, references and language tools specific to the language to be localized. If these materials do not already exist, they must be created.

Wherever possible, information on FOSS localization should be shared with the international FOSS development community so that the necessary tools do not have to be recreated by every team.

Costs of FOSS Localization

Technically speaking, localizing FOSS costs about as much as localizing commercial software. Only the techniques of programming are significantly different, since the linguistic and operational challenges exist no matter what type of software is to be localized. To localize any software, the following are needed:

- Office space.
- Office equipment and tools.
- Technical staff.
- Access to technical information.
- Access to linguists and translators.

The largest cost will be staff salaries. The total cost of a project depends heavily on the wage expectations of local technical, translation, writing and testing staff, and their individual levels of experience with software localization, language and cultural issues.

The programmers and project managers probably require a higher-than-average education and salary, but most of the other staff utilizes skills that are not particular to software and can be found more readily in the general population.

Trainers are hired when the software is near finalization, and presumably remain employed in teaching new users, system administrators and developers how to use the software.

For countries seeking independence from proprietary English language software, a permanent local office whose purpose is to train and disseminate technical information about localization could yield exponential savings. This establishment could be associated with a public library or university, where interested parties can access information at little or no cost.

FOSS can often operate well on older computers. This offers advantages to both developed countries with an overstock of used computers they must dispose of, and developing countries that can configure these computers to operate FOSS in the local language.

The total cost of localizing any particular piece of software is highly variable. Each project requires individual analysis for complexity, experience and availability of technical staff, and the characteristics of the local language.

Software cost and schedule estimating is not a simple calculation. In addition to a rough estimate based on the number of message strings to be translated, other factors must be considered.

- **Experience**: Do the programmers, translators and testers have previous experience with this kind of work? If not, it will require extra time and effort to train them in the processes and standards of localization. But translators learn very quickly, and productivity increases dramatically after the first month or two. With a stable team, the members become very productive.

- **Environment**: Does the staff have the tools and equipment needed to perform the work in a professional manner? Without modern office space, tools and techniques, it is unrealistic to expect the staff to perform at top efficiency.

- **Linguistic factors**: How different is the local language from English? Translating from English to Swedish, for example, is fairly simple. The grammar, length of words and vocabulary is very similar. There is near universal fluency in English, and translators are easy to find. On the other hand, translating from English to Lao is very difficult. The grammar, spelling conventions, word length, collation, and other factors are not similar at all. So the size and position of user interface elements must be changed. In addition, a lack of experienced translators or even of a basic technical glossary means that projects would begin from practically nothing and take much more time and effort.

- **Scope**: How much is enough? Is it acceptable to merely change the primary user interface menus and commands? Should the help files also be translated? What about documentation and user training materials? Are ''anglicisms''acceptable? How many new words will be introduced into the language? To avoid failure, a very clear definition of the project's scope is necessary.

- **Metrics**: Professional software cost and estimating relies on the experience of previous projects for determining future schedules. If there is little heuristic evidence to rely on for estimating, the first few project estimates can only be educated guesses. After several projects have been completed, the actual time for completion can be compared to the initial estimates in order to refine future estimates. So it is important that accurate records of person hours, resources, experience and other factors are collected for future reference.

With the points mentioned above in mind, consider the following formula as a very rough "rule of thumb" for estimating localization project schedules.

Estimating Localization Project Schedules: Rule of Thumb

$$\frac{\left[\text{Number of message strings to be translated}\right] \times \left[\text{Estimated time to translate an "average" string in minutes}\right] \times \left[\text{Level of experience of translators}\right]^{\bullet}}{\left[60\right]^{*} \times \left[\text{"Actual" person hours per week}\right]^{+} \times \left[\text{Number of staff available}\right]} + \left[\text{10-20\% for management and training}\right] = \text{Estimated weeks to completion}$$

- **less experience = more time (Scale 1.5 = less experience, and .75 = experienced)**

*** to find person hours**

+ always less than 40 per week; usually around 20 per week

Note: If no English/local language technical dictionary is available, it must be created before the work can begin. This is a separate project. Once the technical dictionary is completed, it must then be entered into a Translation Memory database (such as KBabel) to allow for consistency in the translation. Without these essential tools, no software can realistically be localized.

Example Case 1:

1. 10,000 message strings to be translated
2. 10 minutes per message string
3. Less than average experience with software localization translating tools and processes.
4. 10 staff members

Example Case 1 Estimate:

1. 10,000 x 10 minutes (divided by 60) = 1,666 person-hours
2. 20 "actual" man hours per week = 83.33 person-weeks
3. Add 16.66 man weeks for testing and editing = 99.99 person-weeks
4. Add 16.66 man weeks for management and training = 116.65 person-weeks
5. Multiply by 1.5 to reflect lack of experience = 174.97 person-weeks
6. Divide by 10 staff members = **17.4975 weeks**

In other words, such a project would require a staff of 10 people working almost five months. If the average salary for these professionals is a thousand dollars a month, costs for staff alone is USD50,000.00.

Add 10 computers, office space, Internet connectivity, copy machines and other routine expenses, and a rough estimate of the overall cost of localization can be made.

Consider the same example with the following change: average experience with software localization translating tools and processes.

1. Multiply by **1.0** to reflect average level of experience = 116.65 person-weeks
2. Divide by 10 staff members = **11.665 weeks**.

A project with experienced staff would require less than four months. If the average salary for these professionals is a thousand dollars a month, costs for staff alone is USD40,000.00.

Consider the same example with the following change: staff with more experience with software localization, translation tools and processes.

1. Multiply by **.75** to reflect above average experience = 81.24 person-weeks
2. Divide by 10 staff members = **8.124 weeks.**

An above average staff of 10 people, requiring no additional training, will need only about two months. If the average salary for these professionals is a thousand dollars a month, cost for staff is only USD20,000.00. Compared to proprietary products, FOSS is ideal for localizing. When a few projects have been completed, the costs drop quickly because the underlying concepts and techniques remain the same. The first few projects must develop new dictionaries, tools and specialties relating to language and technical processes. The experience of the staff is the key to increased productivity.

Once these are in place and well understood, the time and money required to complete additional projects are reduced. Because FOSS developers tend to adhere to open standards, the developers of a local version do not have to reverse-engineer the code to guess what work must be done. The localization process should be very similar from project to project. With commercial proprietary closed software, the opposite is true.

As we have seen, a major pitfall of proprietary software is that only the owner of the copyright can maintain or modify it. With FOSS, any person with the appropriate skills can do the work. So, instead of users being locked into an expensive maintenance contract with a single foreign vendor, support and maintenance of software can be freely contracted to a wide variety of local companies.

Programmers working alone cannot localize software. When estimating the cost, time and effort required for any localization, set aside only about 10 percent of the budget for technical issues. All the rest goes to the time-consuming task of translating, writing, testing and training.

The Work of Localization

When a proprietary company localizes software, it first determines whether the effort would be commercially viable. Then it hires localization experts, including linguists and cultural experts, to develop a technical dictionary. Meanwhile, well-paid analysts and programmers modify the software to accept and display the script for the language.

The lion's share of localization work involves translating and then replacing the labels, menu items, error messages and help files of the software. Sometimes the appearance of the software must also be modified to fit awkwardly long or short words. The technical dictionary, in some cases, is the first of its kind for that language, and new terms are invented.

Closely following the technical dictionary and programming standards, teams of technical writers enter the new phrases alongside the English original. When it is all done, testers ensure not only that every message has been translated, but also that the terminology is consistent and logical.

Work of this sort follows an exponential curve, where the initial work is painfully slow, and then accelerates rapidly when the technical dictionary and standards are well established. After a few programs have been localized, an experienced team can localize additional software at greatly reduced costs. Marketing and training determine how successful they are in getting users to adopt the software. Often, they publish their technical dictionaries and standards, selling them for a profit or making them available free of charge to governments, universities and the online community.

Language Considerations

FOSS has typically been localized by a few volunteers working remotely, without the benefit of linguists or a technical dictionary for translation. The work can take a long time and it can be riddled with inconsistencies or errors.

The pace of FOSS localization is uneven. In countries where the language is similar to English and there are many bilingual volunteers, FOSS localization is well established. Where governments and other agencies have stepped in to provide financial support for localization, the results have also been impressive. (The "CJK" partnership of China, Japan, and Korea stands out as an example.)

In countries without much technical infrastructure, localization of both commercial software and FOSS is slow. It is far slower when the language is not of the Indo-European group. Commercial companies see little profit in the work, and few local professionals have the time or skills to localize FOSS. Even though the source code is freely available for localization work to begin, few specialized technical standards or technical dictionaries exist.

Some languages, particularly those with Latin-based scripts, are relatively easy to localize. Others can be very difficult. As an example, both Lao and Thai share a 42-consonant script, with vowel and intonation marks. These scripts follow complex rules of layout involving consonants, vowels, special symbols, conjuncts and ligatures. All of these writing systems share certain characteristics: spaces are not necessarily used to separate words, and vowels appear before and after, under, over, and after consonants.

Thai and Lao volunteers responsible for localizing FOSS have saved a great deal of time and avoided frustration by cooperating on technical issues, and sharing information on resources and tools.

Across Asia, opportunities exist for shared localization efforts at the inter-governmental level. Many other Asian languages share similarities, and often the programming tasks are nearly identical across similar language groups. Properly funded and organized, pan-Asian software localization is a realistic goal.

ANNEX A. LOCALIZATION – KEY CONCEPTS

This annex provides a quick tour of the key concepts of localization, so that those interested in localizing FOSS for their own language, get a broad picture of the kind of knowledge that is needed. The next annex provides the technical details required to get started.

Standardization

When two or more entities interact, common conventions are important. Car drivers must abide by traffic rules to prevent accidents. People need common conventions on languages and gestures to communicate. Likewise, software needs standards and protocols to interoperate seamlessly. In terms of software engineering, contracts between parts of programs need to be established before implementation. The contracts are most important for systems developed by a large group of individual developers from different backgrounds, and are extremely essential for cross-platform interoperability.

Standards provide such contracts for all computing systems in the world. Software developers need to conform to such conventions to prevent miscommunication. Therefore, standardization should be the very first step for any kind of software development, including localization.

To start localization, it is a good idea to study related standards and use them throughout the project. Nowadays, many international standards and specifications have been developed to cover the languages of the world. If these do not fit the project's needs, one may consider participating in standardization activities. Important sources are:

- *ISO/IEC JTC1 (International Organization for Standardization and International Electrotechnical Commission Joint Technical Committee 1):* A joint technical committee for international standards for information technology. There are many subcommittees (SC) for different categories, under which working groups (WG) are formed to work on subcategories of standards. For example, ISO/IEC JTC1/SC2/WG2 is the working group for Universal Coded Character Set (UCS). The standardization process, however, proceeds in a closed manner. If the national standard body is an ISO/IEC member, it can propose the requirements for the project. Otherwise, one may need to approach individual committees. They may ask for participation as a specialist. Information for JTC1/SC2 (coded character sets) is published at *anubis.dkuug.dk/JTC1/SC2*. Information for JTC1/SC22 (programming languages, their environments and system software interfaces) is at *anubis.dkuug.dk/JTC1/SC22.*

- *Unicode Consortium:* A non-profit organization working on a universal character set. It is closely related to ISO/IEC JTC1 subcommittees. Its Web site is at *www.unicode.org,* where channels of contribution are provided.

- *Free Standards Group:* A non-profit organization dedicated to accelerating the use of FOSS by developing and promoting standards. Its Web site is at *www.freestandards.org.* It is open to participation. There are a number of work groups under its umbrella, including OpenI18N for internationalization (*www.openi18n.org*).

Note, however, that some issues such as national keyboard maps and input/output methods are not covered by the standards mentioned above. The national standards body should define these standards, or unify existing solutions used by different vendors, so that users can benefit from the consistency.

Unicode

Characters are the most fundamental units for representing text data of any particular language. In mathematical terms, the *character set* defines the set of all characters used in a language. In ICT terms, the character set must be *encoded* as bytes in the storage, according to some conventions, called *encoding.* These conventions must be agreed upon both by the sender and receiver of data for the information to remain intact and exact.

In the 1970s, the character set used by most programs consisted of letters of the English alphabet, decimal digits and some punctuation marks. The most widely used encoding was the 7-bit ASCII (American

Standard Code for Information Interchange), in which up to 128 characters can be represented, which is just sufficient for English. However, when the need to use non-English languages in computers arose, other encodings were defined. The concept of *codepages* was devised as enhancements to ASCII by adding characters as the second 7-bit half, making an 8-bit code table in total. Several codepages were defined by vendors for special characters for decoration purpose and for Latin accents. Some non-European languages were added by this strategy, such as Hebrew and Thai. National standards were defined for character encoding.

The traditional encoding systems were not suitable for Asian languages that have large character sets and particular complexities. For example, the encoding of Han characters used by the Chinese, Japanese and Korean (CJK), the total number of which are still not determined, is much more complicated. A large number of codepages must be defined to cover all of them. Moreover, compatibility with other single-byte encodings is another significant challenge. This ends up in some multi-byte encodings for CJK.

However, having a lot of encoding standards to support is a problem for software developers. A group of vendors thus agreed to work together to define a single character set that covers the characters of all languages of the world, so that developers have a single point of reference, and users have a single encoding. The Unicode Consortium was thus founded. Major languages in the world were added to the code table. Later on, ISO and IEC formed JTC1/SC2/WG2 to standardize the code table, which is published as ISO/IEC 10646. Unicode is also a member of the working group, along with standard bodies of ISO member countries. Both Unicode and ISO/IEC 10646 are synchronized, so the code tables are the same. But Unicode also provides additional implementation guidelines, such as character properties, rendering, editing, string collation, etc.

Nowadays, many applications have moved to Unicode and have benefited from the clear definitions for supporting new languages. Users of Unicode are able to exchange information in their own languages, especially through the Internet, without compatibility issues.

Fonts

Once the character set and encoding of a script are defined, the first step to enabling it on a system is to display it. Rendering text on the screen requires some resource to describe the shapes of the characters, i.e., the *fonts,* and some process to render the character images as per script conventions. The process is called the *output method.* This section will try to cover important aspects of these requirements.

Characters and Glyphs

A *font* is a set of *glyphs* for a character set. A *glyph* is an appearance form of a character or a sequence of characters. It is quite important to distinguish the concepts of characters and glyphs. For some scripts, a character can have more than one variation, depending on the context. In that case, the font may contain more than one glyph for each of those characters, so that the text renderer can dynamically pick the appropriate one. On the other hand, the concept of *ligatures,* such as "ff" in English text, also allows some sequence of characters to be drawn together. This introduces another kind of mapping of multiple characters to a single glyph.

Bitmap and Vector Fonts

In principle, there are two methods of describing glyphs in fonts: bitmaps and vectors. Bitmap fonts describe glyph shapes by plotting the pixels directly onto a two-dimensional grid of determined size, while vector fonts describe the outlines of the glyphs with line and curve drawing instructions. In other words, bitmap fonts are designed for a particular size, while vector fonts are designed for all sizes. The quality of the glyphs rendered from bit-map fonts always drops when they are scaled up, while that from vector fonts does not. However, vector fonts often render poorly in small sizes in low-resolution devices, such as computer screens, due to the limited pixels available to fit the curves. In this case, bitmap fonts may be more precise.

Nevertheless, the quality problem at low resolution has been addressed by font technology. For example:

- *Hinting,* additional guideline information stored in the fonts for rasterizers to fit the curves in a way that preserves the proper glyph shape.

- *Anti-aliasing,* capability of the rasterizer to simulate unfitted pixels with some illusion to human perception, such as using grayscales and coloured-subpixels, resulting in the feeling of "smooth curves."

These can improve the quality of vector fonts at small sizes. Moreover, the need for bitmap fonts in modern desktops is gradually diminishing.

Font Formats

Currently, the X Window system for GNU/Linux desktop supports many font formats.

BDF Fonts

BDF (Bit-map Distribution Format) is a bitmap font format of the X Consortium for exchanging fonts in a form that is both human-readable and machine-readable. Its content is actually in plain text.

PCF Fonts

PCF (Portable Compiled Format) is just the compiled form of the BDF format. It is binary and thus, only machine-readable. The utility that compiles BDF into PCF is *bdftopcf.* Although BDF fonts can be directly installed into the X Window system, they are usually compiled for better performance.

Type 1 Fonts

Type 1 is a vector font standard devised by Adobe and supported by its *Postscript* standard. So it is well supported under most UNIX and GNU/Linux, through the X Window system and Ghostscript. Therefore, it is the recommended format for traditional UNIX printing.

TrueType Fonts

TrueType is a vector font standard developed by Apple, and is also used in Microsoft Windows. Its popularity has grown along with the growth of Windows. XFree86 also supports TrueType fonts with the help of the FreeType library. Ghostscript has also supported TrueType. Thus, it becomes another potential choice for fonts on GNU/Linux desktops.

OpenType Fonts

Recently, Adobe and Microsoft have agreed to create a new font standard that covers both Type 1 and TrueType technologies with some enhancements to cover the requirements of different scripts in the world. The result is OpenType.

An OpenType font can describe glyph outlines with either Type 1 or TrueType splines. In addition, information for relative glyph positioning (namely, GPOS table) has been added for combining marks to base characters or to other marks, as well as some glyph substitution rules (namely, GSUB table), so that it is flexible enough to draw characters of various languages.

Output Methods

Output method is a procedure for drawing texts on output devices. It converts text strings into sequences of properly positioned glyphs of the given fonts. For the simple cases like English, the character-to-glyph mapping may be straightforward. But for other scripts the output methods are more complicated. Some could be with combining marks, some written in directions other than left-to-right, some with glyph variations of a single character, some requiring character reordering, and so on.

With traditional font technologies, the information for handling complex scripts is not stored in the fonts. So the output methods bear the burden. But with OpenType fonts, where all of the rules are stored, the output methods just need the capability to read and apply the rules.

Output methods are defined at different implementations. For X Window, it is called *X Output Method (XOM).* For GTK+, it uses a separate module called *Pango.* For Qt, it implements the output method by some classes. Modern rendering engines are now capable of using OpenType fonts. So, there are two ways of drawing texts in output method implementations. If you are using TrueType or Type 1 fonts and your script has some complications over Latin-based languages, you need to provide an output method that knows how to process and typeset characters of your script. Otherwise, you may use OpenType fonts with OpenType tables that describe rules for glyph substitution and positioning.

Input Methods

There are many factors in the design and implementation of input methods. The more different the character set size and the input device capability are, the more complicated the input method becomes. For example, inputting English characters with a 104-key keyboard is straightforward (mostly one-to-one – that is, one key stroke produces one character), while inputting English with mobile phone keypad requires some more steps. For languages with huge character sets, such as CJK, character input is very complicated, even with PC keyboards.

Therefore, analysis and design are important stages of input method creation. The first step is to list all the characters (not glyphs) needed for input, including digits and punctuation marks. The next step is to decide whether it can be matched one-to-one with the available keys, or whether it needs some composing (like European accents) or conversion (like CJK Romanji input) mechanisms in which multiple key strokes are required to input some characters.

When the input scheme is decided for the script, the keyboard layout may be designed. Good keyboard layout should help users by putting most frequently used characters in the home row, and the rest in the upper and lower rows. If the script has no concept of upper/lower cases (which is almost the case for non-Latin scripts), rare characters may be put in the shift positions.

Then, there are two major steps to implement the input method. First, a map of the keyboard layout is created. This is usually an easy step, as there are existing keyboard maps to refer to. Then, if necessary, the second step is to write the input method based on the keyboard map. In general, this means writing an input method module to plug into the system framework.

Locales

Locale is a term introduced by the concept of *internationalization (I18N),* in which generic frameworks are made so that the software can adjust its behaviour to the requirements of different native languages, cultural conventions and coded character sets, without modification or re-compilation.

Within such frameworks, *locales* are defined for describing particular cultures. Users can configure their systems to pick up their locales. The programs will load the corresponding predefined *locale definition* to accomplish internationalized functions. Therefore, to make internationalized software support a new language or culture, one must create a *locale definition* and fill up the required information, and things will work without having to touch the software code.

According to POSIX,[1] a number of C library functions, such as date and time formats, string collation, numerical and monetary formats, are locale-dependent. ISO/IEC 14652 has added more features to POSIX locale specifications and defined new categories for paper size, measurement unit, address and telephone formats, and personal names. GNU C library has implemented all of these categories. Thus, cultural conventions may be described through it.

Locale definitions are discussed in detail on pages 41-42.

Translation

Translating messages in programs, including menus, dialog boxes, button labels, error messages, and so on, ensures that local users, not familiar with English, can use the software. This task can be accomplished only after the input methods, output methods and fonts are done – or the translated messages will become useless.

There are many message translation frameworks available, but the general concepts are the same. Messages are extracted into a working file to be translated and compiled into a hash table. When the program executes, it loads the appropriate translation data as per locale. Then, messages are quickly looked up for the translation to be used in the user interface.

Translation is a labour-intensive task. It takes time to translate a huge number of messages, which is why it is always done by a group of people. When forming a team, make sure that all members use consistent terminology in all parts of the programs. Therefore it is vital to work together in a forum through close

[1]POSIX is the acronym for Portable Operating System specification

discussion and to build the glossary database from the decisions made collectively. Sometimes the translator needs to run the program to see the context surrounding the message, in order to find a proper translation. At other times the translator needs to investigate the source code to locate conditional messages, such as error messages. Translating each message individually in a literal manner, without running the program, can often result in incomprehensible outputs.

Like other FOSS development activities, translation is a long-term commitment. New messages are usually introduced in every new version. Even though all messages have been completed in the current version, it is necessary to check for new messages before the next release. There is usually a string freeze period before a version is released, when no new strings are allowed in the code base, and an appropriate time period is allocated for the translators. Technical aspects of the message translation process are discussed on page 45.

GNU/Linux Desktop Structure

Before planning to enable a language in GNU/Linux desktop, a clear understanding of the overview of its structure is required. GNU/Linux desktop is composed of layers of subsystems working on top of one another. Every layer has its own locale-dependent operations. Therefore, to enable a language completely, it is necessary to work in all layers. The layers, from the bottom up, are as follow (See Figure 1):

1. *The C Library.* C is the programming language of the lowest level for developing GNU/Linux applications. Other languages rely on the C library to make calls to the operating system kernel.

2. *The X Window.* In most UNIX systems, the graphical environment is provided by the X Window system. It is a client-server system, where X clients make requests to X server and receive events from it across the network connection (or through a local inter-process communication channel) based on *X protocol.* A library called *X Library (Xlib)* encapsulates this protocol by a set of application programming interfaces (API), so that X clients can do everything in terms of function calls. Due to its liberal license terms, which allow even commercial redistributions, there have been several versions of X Window in the UNIX market. For GNU/Linux, XFree86 is the major code base, although new releases of some major distributions are now migrating to the newly forked X.org released in April 2004. All forks differ mainly in X server implementation and some extensions. But the X protocol and Xlib function calls are still standardized.

3. *Toolkits.* Writing programs using the low-level Xlib can be tedious as well as a source of inconsistent GUI when all applications draw menus and buttons by their own preferences. Some

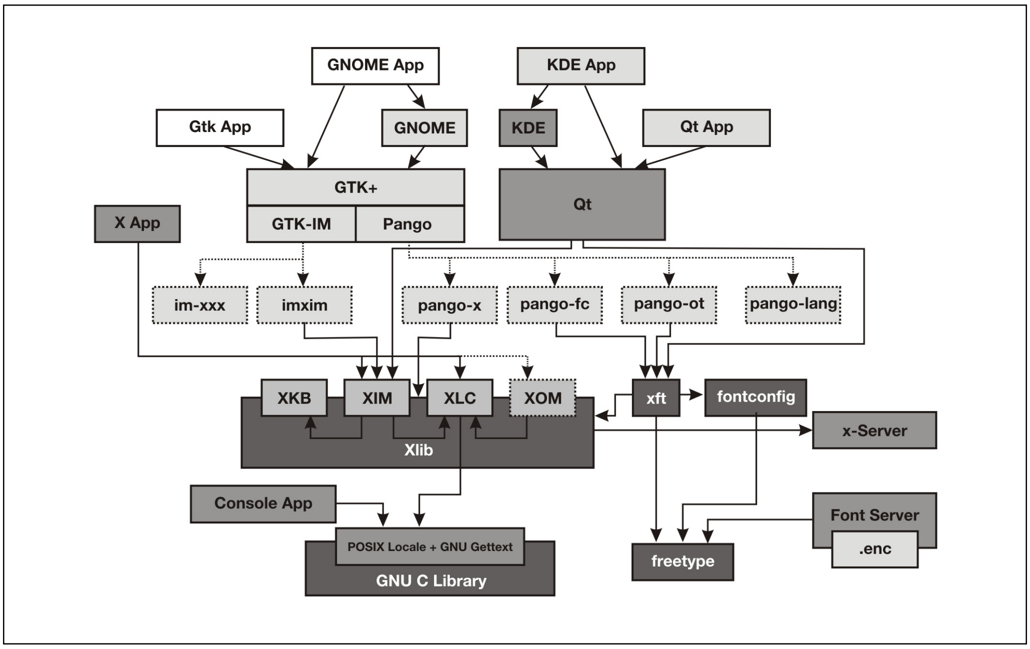


Figure 1 GNU/Linux Desktop Structure and Internationalization

libraries are developed as a middle layer to help reduce both problems. In X terminology, these libraries are called *toolkits.* And the GUI components they provide, such as buttons, text entries, etc., are called *widgets.* Many historic toolkits have been developed in the past, either by the X Consortium itself like the X Toolkit and Athena widget set (Xaw), or by vendors like XView from Sun, Motif from Open Group, etc. In the FOSS realm, the toolkits most widely adopted are *GTK+ (The GIMP Toolkit)*[2] and *Qt.*[3]

4. *Desktop Environments.* Toolkits help developers create a consistent look-and-feel among a set of programs. But to make a complete desktop, applications need to interoperate more closely to form a convenient workplace. The concept of *desktop environment* has been invented to provide common conventions, resource sharing and communication among applications. The first desktop environment ever created on UNIX platforms was *CDE (Common Desktop Environment)* by Open Group, based on its Motif toolkit. But it is proprietary. The first FOSS desktop environment for GNU/Linux is *KDE (K Desktop Environment),*[4] based on TrollTech's Qt toolkit. However, due to some licensing conditions of Qt at that time, some developers didn't like it. A second one was thus created, called *GNOME (GNU Network Object Modelling Environment),*[5] based on GTK+. Nowadays, although the licensing issue of Qt has been resolved, GNOME continues to grow and get more support from vendors and the community. KDE and GNOME have thus become the desktops most widely used on GNU/Linux and other FOSS operating systems such as FreeBSD.

Each component is internationalized, allowing local implementation for different locales:

1. *GNU C Library:* Internationalized according to POSIX and ISO/IEC 14652.

2. *XFree86 (and X Window in general):* Internationalization in this layer includes *X locale (XLC)* describing font set and character code conversion; *X Input Method (XIM)* for text input process, in which *X Keyboard Extension (XKB)* is used in describing keyboard map; and *X Output Method (XOM)* for text rendering. For XOM, it was implemented too late, when both GTK+ and Qt had already handled the rendering by their own solutions. Therefore, it is questionable whether XOM is still needed.

3. *GTK+:* For GTK+ 2, internationalization frameworks have been defined in a modular way. It has its own input method framework called GTK+ IM, where input method modules can be dynamically plugged in as per user command. Text rendering in GTK+ 2 is handled by a separate general-purpose text layout engine called *Pango.* Pango can be used for any application that needs to render multilingual texts and not just for GTK.

4. *Qt:* Internationalization in Qt 3 is done in a minimal way. It relies solely on XIM for all text inputs, and handles text rendering with *QComplexText* C++ class, which relies completely on Unicode data for character properties from Unicode.org.

   For the desktop environment layer, namely, GNOME and KDE, there is no additional internationalization apart from what is provided by GTK+ and Qt.

[2]GTK+,'GTK+ – The GIMP Toolkit'; available from *www.gtk.org.*

[3]TrollTech,'TrollTech – The Creator of Qt – The multi-platform C++ GUI/API'; available from *www.trolltech.com.*

[4]KDE, 'KDE Homepage – Conquer your Desktop!'; available from *www.kde.org.*

[5]GNOME, 'GNOME: The Free Software Desktop Project'; available from *www.gnome.org.*

ANNEX B. LOCALIZATION – TECHNICAL ASPECTS

In this annex, more technical details will be discussed. The aim is to give implementers necessary information to start localization. However, this is not intended to be a hands-on cookbook.

Unicode

As a universal character set that includes all characters of the world, Unicode assigns code points to its characters by 16-bit integers, which means that up to 65,536 characters can be encoded. However, due to the huge set of CJK characters, this has become insufficient, and Unicode 3.0 has extended the index to 21 bits, which will support up to 1,114,112 characters.

Planes

Unicode code point is a numeric value between 0 and 10FFFF, divided into *planes* of 64K characters. In Unicode 4.0, allocated planes are Plane 0, 1, 2 and 14.

Plane 0, ranging from 0000 to FFFF, is called *Basic Multilingual Plane (BMP),* which is the set of characters assigned by the previous 16-bit scheme.

Plane 1, ranging from 10000 to 1FFFF and called *Supplementary Multilingual Plane (SMP),* is dedicated to lesser used historic scripts, special-purpose invented scripts and special notations. These include Gothic, Shavian and musical symbols. Many more historic scripts may be encoded in this plane in the future.

Plane 2, ranging from 20000 to 2FFFF and called *Supplementary Ideographic Plane (SIP),* is the spillover allocation area for those CJK characters that cannot fit into the blocks for common CJK characters in the BMP. Plane 14, ranging from E0000 to EFFFF and called *Supplementary Special-purpose Plane (SSP),* is for some control characters that do not fit into the small areas allocated in the BMP.

There are two more reserved planes Plane 15 and Plane 16, for private use, where no code point is assigned.

Basic Multilingual Plane

Basic Multilingual Plane (BMP), or Plane 0, is most commonly in general documents. Code points are allocated for common characters in contemporary scripts with exactly the same set as ISO/IEC 10646-1, as summarized in Figure 2 in section ý0 Note that the code points between E000 and F900 are reserved for the vendors' private use. No character is assigned in this area.

Character Encoding

There are several ways of encoding Unicode strings for information interchange. One may simply represent each character using a fixed size integer (called *wide char*), which is defined by ISO/IEC 10646 as UCS-2 and UCS-4, where 2-byte and 4-byte integers are used, respectively[6] and where UCS-2 is for BMP only. But the common practice is to encode the characters using variable-length sequences of integers called UTF-8, UTF-16 and UTF-32 for 8-bit, 16-bit and 32-bit integers, respectively.[7] There is also UTF-7 for e-mail transmissions that are 7-bit strict, but UTF-8 is safe in most cases.

UTF-32

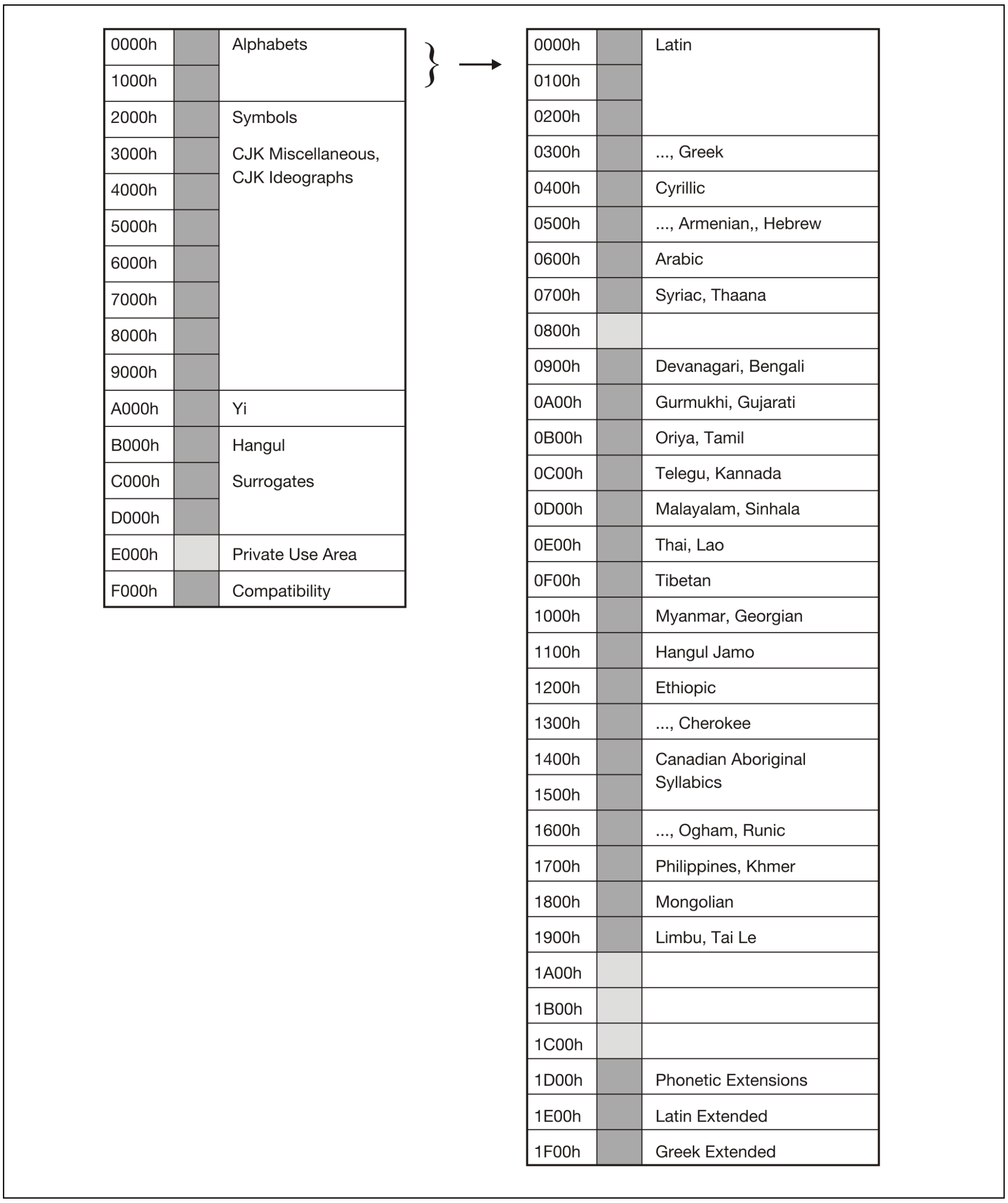
UTF-32 is the simplest Unicode encoding form. Each Unicode code point is represented directly by a single 32-bit unsigned integer. It is therefore, a fixed-width character encoding form. This makes UTF-32 an ideal form for APIs that pass single character values. However, it is inefficient in terms of storage for Unicode strings.

UTF-16

UTF-16 encodes code points in the range 0000 to FFFF (i.e. BMP) as a single 16-bit unsigned integer.

[6]UCS is the acronym for Universal multi-octet coded Character Set

[7]UTF is the acronym for Unicode (UCS) Transformation Format



0000h	■	Alphabets
1000h	■	
2000h	■	Symbols
3000h	■	CJK Miscellaneous, CJK Ideographs
4000h	■	
5000h	■	
6000h	■	
7000h	■	
8000h	■	
9000h	■	
A000h	■	Yi
B000h	■	Hangul
C000h	■	Surrogates
D000h	■	
E000h	□	Private Use Area
F000h	■	Compatibility

$\xrightarrow{}$

0000h	■	Latin
0100h	■	
0200h	■	
0300h	■	..., Greek
0400h	■	Cyrillic
0500h	■	..., Armenian,, Hebrew
0600h	■	Arabic
0700h	■	Syriac, Thaana
0800h	□	
0900h	■	Devanagari, Bengali
0A00h	■	Gurmukhi, Gujarati
0B00h	■	Oriya, Tamil
0C00h	■	Telegu, Kannada
0D00h	■	Malayalam, Sinhala
0E00h	■	Thai, Lao
0F00h	■	Tibetan
1000h	■	Myanmar, Georgian
1100h	■	Hangul Jamo
1200h	■	Ethiopic
1300h	■	..., Cherokee
1400h	■	Canadian Aboriginal Syllabics
1500h	■	
1600h	■	..., Ogham, Runic
1700h	■	Philippines, Khmer
1800h	■	Mongolian
1900h	■	Limbu, Tai Le
1A00h	□	
1B00h	□	
1C00h	□	
1D00h	■	Phonetic Extensions
1E00h	■	Latin Extended
1F00h	■	Greek Extended

Figure 2 Unicode Basic Multilingual Plane

Code points in supplementary planes are instead represented as pairs of 16-bit unsigned integers. These pairs of code units are called *surrogate pairs.* The values used for the surrogate pairs are in the range D800 – DFFF, which are not assigned to any character. So, UTF-16 readers can easily distinguish between single code unit and surrogate pairs. The Unicode Standard[8] provides more details of surrogates.

UTF-16 is a good choice for keeping general Unicode strings, as it is optimized for characters in BMP, which is used in 99 percent of Unicode texts. It consumes about half of the storage required by UTF-32.

UTF-8

To meet the requirements of legacy byte-oriented ASCII-based systems, UTF-8 is defined as variable-width encoding form that preserves ASCII compatibility. It uses one to four 8-bit code units to represent a Unicode character, depending on the code point value. The code points between 0000 and 007F are encoded in a single byte, making any ASCII string a valid UTF-8. Beyond the ASCII range of Unicode, some non-ideographic characters between 0080 and 07FF are encoded with two bytes. Then, Indic scripts and CJK ideographs between 0800 and FFFF are encoded with three bytes. Supplementary characters

[8]The Unicode Consortium. *The Unicode Standard, Version 4.0.*, pp. 76–77.

beyond BMP require four bytes. The Unicode Standard[9] provides more detail of UTF-8.

UTF-8 is typically the preferred encoding form for the Internet. The ASCII compatibility helps a lot in migration from old systems. UTF-8 also has the advantage of being byte-serialized and friendly to C or other programming languages APIs. For example, the traditional string collation using byte-wise comparison works with UTF-8.

In short, UTF-8 is the most widely adopted encoding form of Unicode.

Character Properties

In addition to code points, Unicode also provides a database of character properties called the *Unicode Character Database (UCD)*,[10] which consists of a set of files describing the following properties:

- Name.
- General category (classification as letters, numbers, symbols, punctuation, etc.).
- Other important general characteristics (white space, dash, ideographic, alphabetic, non character, deprecated, etc.).
- Character shaping (bidi category, shaping, mirroring, width, etc.).
- Case (upper, lower, title, folding; both simple and full).
- Numeric values and types (for digits).
- Script and block.
- Normalization properties (decompositions, decomposition type, canonical combining class, composition exclusions, etc.).
- Age (version of the standard in which the code point was first designated).
- Boundaries (grapheme cluster, word, line and sentence).
- Standardized variants.

The database is useful for Unicode implementation in general. It is available at the Unicode.org Web site. The Unicode Standard[11] provides more details of the database.

Technical Reports

In addition to the code points, encoding forms and character properties, Unicode also provides some technical reports that can serve as implementation guidelines. Some of these reports have been included as annexes to the Unicode standard, and some are published individually as Technical Standards.

In Unicode 4.0, the standard annexes are:

- *UAX 9: The Bidirectional Algorithm*
  Specifications for the positioning of characters flowing from right to left, such as Arabic or Hebrew.
- *UAX 11: East-Asian Width*
  Specifications of an informative property of Unicode characters that is useful when interoperating with East-Asian Legacy character sets.
- *UAX 14: Line Breaking Properties*
  Specification of line breaking properties for Unicode characters as well as a model algorithm for determining line break opportunities.
- *UAX 15: Unicode Normalization Forms*
  Specifications for four normalized forms of Unicode text. With these forms, equivalent text (canonical or compatibility) will have identical binary representations. When implementations keep strings in a normalized form, they can be assured that equivalent strings have a unique binary representation.
- *UAX 24: Script Names*
  Assignment of script names to all Unicode code points. This information is useful in mechanisms such as regular expressions, where it produces much better results than simple matches on block names.
- *UAX 29: Text Boundaries*

[9]The Unicode Consortium. *The Unicode Standard, Version 4.0.*, pp. 77–78.

[10]Ibid., pp. 95–104.

[11]Unicode.org, 'Unicode Technical Reports'; available from *www.unicode.org/reports/index.html.*

Guidelines for determining default boundaries between certain significant text elements: grapheme clusters ("user characters"), words and sentences.

The individual technical standards are:

- *UTS 6: A Standard Compression Scheme for Unicode*
  Specifications of a compression scheme for Unicode and sample implementation.
- *UTS 10: Unicode Collation Algorithm*
  Specifications for how to compare two Unicode strings while conforming to the requirements of the Unicode Standard. The UCA also supplies the Default Unicode Collation Element Table (DUCET) as the data specifying the default collation order for all Unicode characters.
- *UTS 18: Unicode Regular Expression Guidelines*
  Guidelines on how to adapt regular expression engines to use Unicode.

All Unicode Technical Reports are accessible from the Unicode.org web site.[12]

Fonts

Font Development Tools

Some FOSS tools for developing fonts are available. Although not as many as their proprietary counterparts, they are adequate to get the job done, and are continuously being improved. Some interesting examples are:

1. *XmBDFEd.*[13] Developed by Mark Leisher, XmBDFEd is a Motif-based tool for developing BDF fonts. It allows one to edit bit-map glyphs of a font, do some simple transformations on the glyphs, transfer information between different fonts, and so on.

2. *FontForge*[14] *(formerly PfaEdit*[15]*).* Developed by George Williams, FontForge is a tool for developing outline fonts, including Postscript Type1, TrueType, and OpenType. Scanned images of letters can be imported and their outline vectors automatically traced. The splines can be edited, and transformations like skewing, scaling, rotating, thickening may be applied and much more. It provides sufficient functionalities for editing Type1 and TrueType fonts properties. OpenType tables can also be edited in its recent versions. One weak point, however, is hinting. It guarantees Type1 hints quality, but not for TrueType.

3. *TTX/FontTools.*[16] Just van Rossum's TTX/FontTools is a tool to convert OpenType and TrueType fonts to and from XML. FontTools is a library for manipulating fonts, written in Python. It supports TrueType, OpenType, AFM and, to a certain extent, Type 1 and some Mac-specific formats. It allows one to dump OpenType tables, examine and edit them with XML or plain text editor, and merge them back to the font.

Font Configuration

There have been several font configuration systems available in GNU/Linux desktops. The most fundamental one is the X Window font system itself. But, due to some recent developments, another font configuration called *fontconfig* has been developed to serve some specific requirements of modern desktops. These two font configurations will be discussed briefly.

First, however, let us briefly discuss the X Window architecture, to understand font systems. X Window[17] is a client-server system. *X servers* are the agents that provide service to control hardware devices, such as video cards, monitors, keyboards, mice or tablets, as well as passes user input events from the devices

[12]Unicode.org, 'Unicode Technical Reports'; available from *www.unicode.org/reports/index.html.*

[13]Leisher, M., 'The XmBDFEd Font Editor'; available from *crl.nmsu.edu/~mleisher/xmbdfed.html.*

[14]Williams, G., 'PfaEdit'; available from *pfaedit.sourceforge.net.*

[15]van Rossum, J., S 'TTX/FontTools'; available from *fonttools.sourceforge.net/.*

[16]Note the difference with Microsoft's "Windows" trademark. X Window is without 's'.

[17]Taylor, O., 'Pango'; available from *www.pango.org.*

to the clients. *X clients* are GUI application programs that request X server to draw graphical objects on the screen, and accept user inputs via the events fed by X server. Note that with this architecture, X client and server can be on different machines in the network. In which case, X server is the machine that the user operates with, while X client can be a process running on the same machine or on a remote machine in the network.

In this client-server architecture, fonts are provided on the server side. Thus, installing fonts means configuring X server by installing fonts and registering them to its font path.

However, since X server is sometimes used to provide thin-client access in some deployments, where X server may run on cheap PCs booted by floppy or across network, or even from ROM, font installation on each X server is not always appropriate. Thus, font service has been delegated to a separate service called *X Font Server (XFS).* Another machine in the network can be dedicated for font service so that all X servers can request font information. Therefore, with this structure, an X server may be configured to manage fonts by itself or to use fonts from the font server, or both.

Nevertheless, recent changes in XFree86 have addressed some requirements to manage fonts at the client side. The Xft extension provides anti-aliased glyph images by font information provided by the X client. With this, the Xft extension also provides font management functionality to X clients in its first version. This was later split from Xft2 into a separate library called *fontconfig*. fontconfig is a font management system independent of X, which means it can also apply to non-GUI applications such as printing services. Modern desktops, including KDE 3 and GNOME 2 have adopted fontconfig as their font management systems, and have benefited from closer integration in providing easy font installation process. Moreover, client-side fonts also allow applications to do all glyph manipulations, such as making special effects, while enjoying consistent appearance on the screen and in printed outputs.

The splitting of the X client-server architecture is not standard practice on stand-alone desktops. However, it is important to always keep the split in mind, to enable particular features.

Output Methods

Since the usefulness of XOM is still being questioned, we shall discuss only the output methods already implemented in the two major toolkits: Pango of GTK+ 2 and Qt 3.

Pango Text Layout Engines

Pango ['Pan' means 'all' in English and 'go' means 'language' in Japanese][18] is a multilingual text layout engine designed for quality text typesetting. Although it is the text drawing engine of GTK+, it can also be used outside GTK+ for other purposes, such as printing.[19] This section will provide localizers with a bird's eye view of Pango. The Pango reference manual[20] should be consulted for more detail.

PangoLayout

At a high level, Pango provides the **PangoLayout** class that takes care of typesetting text in a column of given width, as well as other information necessary for editing, such as cursor positions. Its features may be summarized as follows:

Paragraph Properties

- indent
- spacing
- alignment
- justification
- word/character wrapping modes
- tabs

Text Elements

- get lines and their extents
- get runs and their extents
- character search at (x, y) position
- character logical attributes (is line break, is cursor position, etc.)
- cursor movements

[18]Taylor, O., 'Pango – Design'; available from *www.pango.org/design.shtml*.

[19]GNOME Development Site, 'Pango Reference Manual'; available from *developer.gnome.org/doc/API/2.0/pango/*.

[20]This is a very rough classification. Obviously, there are further steps, such as line breaking, alignment and justification. They need not be discussed here, as they go beyond localization.

Text Contents

- plain text
- markup text

Middle-level Processing

Pango also provides access to some middle-level text processing functions, although most clients in general do not use them directly. To gain a brief understanding of Pango internals, some highlights are discussed here.

There are three major steps for text processing in Pango:[21]

- *Itemize.* Breaks input text into chunks (items) of consistent direction and shaping engine. This usually means chunks of text of the same language with the same font. Corresponding shaping and language engines are also associated with the items.
- *Break*. Determines possible line, word and character breaks within the given text item. It calls the language engine of the item (or the default engine based on Unicode data if no language engine exists) to analyze the logical attributes of the characters (is-line-break, is-char-break, etc.).
- *Shape.* Converts the text item into glyphs, with proper positioning. It calls the shaping engine of the item (or the default shaping engine that is currently suitable for European languages) to obtain a glyph string that provides the information required to render the glyphs (code point, width, offsets, etc.).

Pango Engines

Pango engines are implemented in loadable modules that provide entry functions for querying and creating the desired engine. During initialization, Pango queries the list of all engines installed in the memory. Then, when it itemizes input text, it also searches the list for the language and shaping engines available for the script of each item and creates them for association to the relevant text item.

Pango Language Engines

As discussed above, the Pango language engine is called to determine possible break positions in a text item of a certain language. It provides a method to analyze the logical attributes of every character in the text as listed in Table 3.

Table 3 Pango Logical Attributes

Flag	Description
is_line_break	can break line in front of the character
is_mandatory_break	*must* break line in front of the character
is_char_break	can break here when doing character wrap
is_white	is white space character
is_cursor_position	cursor can appear in front of character
is_word_start	is first character in a word
is_word_end	is first non-word character after a word
is_sentence_boundary	is inter-sentence space
is_sentence_start	is first character in a sentence
is_sentence_end	is first non-sentence character after a sentence
backspace_deletes_character	backspace deletes one character, not entire cluster *(new in Pango 1.3.x)*

Pango Shaping Engines

As discussed above, the Pango shaping engine converts characters in a text item in a certain language

[21]Pascal, I., *X Keyboard Extension*; available from *pascal.tsu.ru/en/xkb/*.

into glyphs, and positions them according to the script constraints. It provides a method to convert a given text string into a sequence of glyphs information (glyph code, width and positioning) and a logical map that maps the glyphs back to character positions in the original text. With all the information provided, the text can be properly rendered on output devices, as well as accessed by the cursor despite the difference between logical and rendering order in some scripts like Indic, Hebrew and Arabic.

Qt Text Layout

Qt 3 text rendering is different from that of GTK+/Pango. Instead of modularizing, it handles all complex text rendering in a single class, called *QComplexText,* which is mostly based on the Unicode character database. This is equivalent to the default routines provided by Pango. Due to the incompleteness of the Unicode database, this class sometimes needs extra workarounds to override some values. Developers should examine this class if a script is not rendered properly.

Although relying on the Unicode database appears to be a straightforward method for rendering Unicode texts, this makes the class rigid and error prone. Checking the Qt Web site regularly to find out whether there are bugs in latest versions is advisable. However, a big change has been planned for Qt 4, which is the Scribe text layout engine, similar to Pango for GTK+.

Input Methods

The needs of keyboard maps and input methods have been discussed on page 37. This section will further discuss how to implement them, beginning with keyboard layouts. Pages 37-38 also mentions that XIM is the current basic input method framework for X Window. Only Qt 3 relies on it, while GTK+ 2 defines its own input method framework. Both XIM and GTK+ IM are discussed here.

Keyboard Layouts

The first step to providing text input for a particular language is to prepare the keyboard map. X Window handles the keyboard map using the *X Keyboard (XKB)* extension. When you start an X server on GNU/Linux, a virtual terminal is attached to it in raw mode, so that keyboard events are sent from the kernel without any translation.

The *raw scan code* of the key is then translated into *keycode* according to the keyboard model. For XFree86 on PC, the keycode map is usually "xfree86" as kept under */etc/X11/xkb/keycodes* directory. The keycodes just represent the key positions in symbolic form, for further referencing.

The *keycode* is then translated into a *keyboard symbol (keysym)* according to the specified layout, such as qwerty, dvorak, or a layout for a specific language, chosen from the data under */etc/X11/xkb/symbols* directory. A keysym does not represent a character yet. It requires an input method to translate sequences of key events into characters, which will be described later. For XFree86, all of the above setup is done via the *setxkbmap* command. (Setting up values in */etc/X11/XF86Config* means setting parameters for *setxkbmap* at initial X server startup.) There are many ways of describing the configuration, as explained in Ivan Pascal's XKB explanation.[22] The default method for XFree86 4.x is the "xfree86" rule (XKB rules are kept under */etc/X11/xkb/rules*), with additional parameters:

- *model – pc104, pc105, microsoft, microsoftplus, …*
- *layout – us, dk, ja, lo, th, …*
  (For XFree86 4.0+, up to 64 groups can be provided as part of layout definition)
- *variant* – (mostly for Latins) *nodeadkeys*
- *option* – group switching key, swap caps, LED indicator, etc.
  (See */etc/X11/xkb/rules/xfree86* for all available options.)

For example:

```
   $ setxkbmap us,th -option grp:alt_shift_toggle,grp_led:scroll
```

Sets layout using US symbols as the first group, and Thai symbols as the second group. The *Alt-Shift*

[22] Pascal, I., *X Keyboard Extension*; available from *pascal.tsu.ru/en/xkb/.*

combination is used to toggle between the two groups. Scroll Lock LED will be the group indicator, which will be on when the current group is not the first group, that is, on for Thai, off for US.

You can even mix more than two languages:

```
$ setxkbmap us,th,lo -option grp:alt_shift_toggle,grp_led:scroll
```

This loads trilingual layout. *Alt-Shift* is used to rotate among the three groups; that is, *Alt-RightShift* chooses the next group and *Alt-LeftShift* chooses the previous group. Scroll Lock LED will be on when the Thai or Lao group is active.

The arguments for *setxkbmap* can be specified in */etc/X11/XF86Config* for initialization on X server startup by describing the "InputDevice" section for keyboard, for example:

```
Section "InputDevice"
  Identifier "Generic Keyboard"
  Driver "keyboard"
  Option "CoreKeyboard"
  Option "XkbRules"   "xfree86"
  Option "XkbModel"   "microsoftplus"
  Option "XkbLayout"  "us,th_tis"
  Option "XkbOptions grp:alt_shift_toggle,lv3:switch,grp_led:scroll"
EndSection
```

Notice the last four option lines. They tell *setxkbmap* to use "xfree86" rule, with "microsoftplus" model (with Internet keys), mixed layout of US and Thai TIS-820.2538, and some more options for group toggle key and LED indicator. The "lv3:switch" option is only for keyboard layouts that require a 3rd level of shift (that is, one more than the normal shift keys). In this case for "th_tis" in XFree86 4.4.0, this option sets *RightCtrl* as 3rd level of shift.

Providing a Keyboard Map

If the keyboard map for a language is not available, one needs to prepare a new one. In XKB terms, one needs to prepare a *symbols* map, associating *keysyms* to the available keycodes.

The quickest way to start is to read the available symbols files under the */etc/X11/xkb/symbols* directory. In particular, the files used by default rules of XFree86 4.3.0 are under the *pc/* subdirectory. Here, only one group is defined per file, unlike the old files in its parent directory, in which groups are pre-combined. This is because XFree86 4.3.0 provides a flexible method for mixing keyboard layouts.

Therefore, unless you need to support the old versions of XFree86, all you need to do is to prepare a single-group symbols file under the *pc/* subdirectory.

Here is an excerpt from the *th_tis* symbols file:

```
partial default alphanumeric_keys
xkb_symbols "basic" {
  name[Group1]= "Thai (TIS-820.2538)";
  // The Thai layout defines a second keyboard group and changes
  // the behavior of a few modifier keys.
  key <TLDE> { [ 0x1000e4f,       0x1000e5b ] };
  key <AE01> { [ Thai_baht,       Thai_lakkhangyao] };
  key <AE02> { [ slash,           Thai_leknung ] };
  key <AE03> { [ minus,           Thai_leksong ] };
  key <AE04> { [ Thai_phosamphao, Thai_leksam ] };
  ...
};
```

Each element in the *xkb_symbols* data, except the first one, is the association of keysyms to the keycode for unshift and shift versions, respectively. Here, some keysyms are predefined in Xlib. You can find the complete list in *<X11/keysymdef.h>*. If the keysyms for a language are not defined there, the Unicode keysyms, can be used, as shown in the *<TLDE>* key entry. (In fact, this may be a more effective way for adding new keysyms.) The Unicode value must be prefixed with "0x100" to describe the keysym for a single character.

For more details of the file format, see Ivan Pascal's XKB explanation.[23] When finished, the *symbols.dir* file should be regenerated so that the symbols file is listed:

```
# cd /etc/X11/xkb/symbols
# xkbcomp -lhlpR '*' -o ../symbols.dir
```

Then, the new layout may be tested as described in the previous section.

Additionally, entries may be added to */etc/X11/xkbcomp/rules/xfree86.lst* so that some GUI keyboard configuration tools can see the layout.

Once the new keyboard map is completed, it may also be included in XFree86 source where the data for XKB are kept under the *xc/programs/xkbcomp* subdirectory.

XIM – X Input Method

For some languages, text input is as straightforward as one-to-one mapping from keysyms to characters, such as English. For European languages, this is a little more complicated because of accents. But for Chinese, Japanese and Korean (CJK), the one-to-one mapping is impossible. They require a series of keystroke interpretations to obtain each character.

X Input Method (XIM) is a locale-based framework designed to address the requirements of text input for any language. It is a separate service for handling input events as requested by X clients. Any text entry in X clients is represented by *X Input Context (XIC)*. All the keyboard events will be propagated to the XIM, which determines the appropriate action for the events based on the current state of the XIC, and passes back the resulting characters.

Internally, a common process of every XIM is to translate keyboard scan code into keycode and then to keysym, by calling XKB, whose process detail has been described in previous sections. The following processes to convert keysyms into characters are different for different locales.

In general cases, XIM is usually implemented using the client-server model. More detailed discussion of XIM implementation is beyond the scope of this document. Please see Section 13.5 of the Xlib document[24] and the XIM protocol[25] for more information.

In general, users can choose their favourite XIM server by setting the system environment *XMODIFIERS,* like this:

```
$ export LANG=th_TH.TIS-620
$ export XMODIFIERS="@im=Strict"
```

This specifies *Strict* input method for Thai locale.

GTK+ IM

As a cross-platform toolkit, GTK+ 2 defines its own framework using pure GTK+ APIs, instead of relying on the input methods of each operating system. This provides high-level of abstraction, making input methods development a lot easier than writing XIM servers. In any case, GTK+ can still use the several existing XIM servers through the *imxim* bridging module. Besides, the input methods developed become immediately available to GTK+ in all platforms it supports, including XFree86, Windows, and GNU/Linux framebuffer console. The only drawback is that the input methods cannot be shared with non-GTK+ applications.

[23]Gettys, J., Scheifler, R.W., *'Xlib – C Language X Interface, X Consortium Standard, X Version 11 Release 6.4.'*

[24]Narita, M., Hiura, H., *The Input Method Protocol Version 1.0. X Consortium Standard, X Version 11 Release 6.4*.

[25]OpenI18N.org. *OpenI18N Locale Name Guideline, Version 1.1 – 2003-03-11]*; available from *www.openi18n.org/docs/text/LocNameGuide-V11.txt.*

Client Side

A normal GTK+-based text entry widget will provide an "Input Methods" context menu that can be opened by right clicking within the text area. This menu provides the list of all installed GTK+ IM modules, which the user can choose from. The menu is initialized by querying all installed modules for the engines they provide.

From the client's point of view, each text entry is represented by an *IM context,* which communicates with the IM module after every key press event by calling a key filter function provided by the module. This allows the IM to intercept the key presses and translate them into characters. Non-character keys, such as function keys or control keys, are not usually intercepted. This allows the client to handle special keys, such as shortcuts.

There are also interfaces for the other direction. The IM can also call the client for some actions by emitting GLib signals, for which the handlers may be provided by the client by connecting callbacks to the signals:

- *"preedit_changed"*
  Uncommitted (pre-edit) string is changed. The client may update the display, but not the input buffer, to let the user see the keystrokes.
- *"commit"*
  Some characters are committed from the IM. The committed string is also passed so that the client can take it into its input buffer.
- *"retrieve_surrounding"*
  The IM wants to retrieve some text around the cursor.
- *"delete_surrounding"*
  The IM wants to delete the text around the cursor. The client should delete the text portion around the cursor as requested.

IM Modules

GTK+ input methods are implemented using loadable modules that provide entry functions for querying and creating the desired IM context. These are used as interface with the "Input Methods" context menu in text entry areas.

The IM module defines a new IM context class or classes and provides filter functions to be called by the client upon key press events. It can determine proper action to the key and return TRUE if it means to intercept the event or FALSE to pass the event back to the client.

Some IM (e.g., CJK and European) may do a stateful conversion which is incrementally matching the input string with predefined patterns until each unique pattern is matched before committing the converted string. During the partial matching, the IM emits the *"preedit_changed"* signal to the client for every change, so that it can update the pre-edit string to the display. Finally, to commit characters, the IM emits the *"commit"* signal, along with the converted string as the argument, to the IM context. Some IM (e.g., Thai) is context-sensitive. It needs to retrieve text around the cursor to determine the appropriate action. This can be done through the *"retrieve_surrounding"* signal.

In addition, the IM may request to delete some text from the client's input buffer as required by Thai advanced IM. This is also used to correct the illegal sequences. This can be done via the *"delete_surrounding"* signal.

Locales

As mentioned in earlier, the GNU C library is internationalized according to POSIX and ISO/IEC 14652. Both locales are discussed in this section.

Locale Naming

A locale is described by its language, country and character set. The naming convention as given in OpenI18N guideline[26] is:

lang_territory.codeset[@*modifiers*]

[26]Library of Congress, ISO 639-2 Registration Authority; available from *lcweb.loc.gov/standards/iso639-2*.

where

- *lang* is a two-letter language code defined in ISO 639:1988. Three-letter codes in ISO 639-2 are also allowed in the absence of the two-letter version. The ISO 639-2 Registration Authority at Library of Congress[27] has a complete list of language codes.
- *territory* is a two-letter country code defined in ISO 3166-1:1997. The list of two-letter country codes is available online from ISO 3166 Maintenance agency.[28]
- *codeset* describes the character set used in the locale.
- *modifiers* add more information for the locale by setting options (turn on flags or use equal sign to set values). Options are separated by commas. This part is optional and implementation-dependent. Different I18N frameworks provide different options.

For example:

- fr_CA.ISO-8859-1= French language in Canada using ISO-8859-1 character set
- th_TH.TIS-620 = Thai language in Thailand using TIS-620 encoding

If *territory* or *codeset* is omitted, default values are usually resolved by means of locale aliasing.

Note that for the GNU/Linux desktop, the *modifiers* part is not supported yet. Locale modifiers for X Window are to be set through the XMODIFIERS environment instead.

Character Sets

Character set is part of locale definition. It defines all characters in a character set as well as how they are encoded for information interchange. In the GNU C library (glibc), locales are described in terms of Unicode.

A new character set is described as a Unicode subset, with each element associated by a byte string to be encoded in the target character set. For example, the UTF-8 encoding is described like this:

```
...
<U0041>  /x41           LATIN CAPITAL LETTER A
<U0042>  /x42           LATIN CAPITAL LETTER B
<U0043>  /x43           LATIN CAPITAL LETTER C
...
<U0E01>  /xe0/xb8/x81   THAI CHARACTER KO KAI
<U0E02>  /xe0/xb8/x82   THAI CHARACTER KHO KHAI
<U0E03>  /xe0/xb8/x83   THAI CHARACTER KHO KHUAT
...
```

The first column is the Unicode value. The second is the encoded byte string. And the rest are comments.

As another example, TIS-620 encoding for Thai is simple 8-bit single-byte. The first half of the code table is the same as ASCII, and the second half begins encoding the first character at 0xA1. Therefore, the character map looks like:

```
...
<U0041>  /x41           LATIN CAPITAL LETTER A
<U0042>  /x42           LATIN CAPITAL LETTER B
<U0043>  /x43           LATIN CAPITAL LETTER C
...
<U0E01>  /xa1           THAI CHARACTER KO KAI
<U0E02>  /xa2           THAI CHARACTER KHO KHAI
<U0E03>  /xa3           THAI CHARACTER KHO KHUAT
...
```

[27]ISO, ISO 3166 Maintenance agency (ISO 3166/MA) – ISO's focal point for country codes; available from *www.iso.org/iso/en/prods-services/iso3166ma/index.html.*

[28]ISO/IEC, ISO/IEC JTC1/SC22/WG20 – Internationalization; available from *anubis.dkuug.dk/jtc1/sc22/wg20.*

POSIX Locales

According to POSIX, standard C library functions are internationalized according to the following categories:

Category	**Description**
`LC_CTYPE`	character classification
`LC_COLLATE`	string collation
`LC_TIME`	date and time format
`LC_NUMERIC`	number format
`LC_MONETARY`	currency format
`LC_MESSAGES`	messages in locale language

Setting Locale

A C application can set current locale with the *setlocale()* function (declared in <*locale.h*>). The first argument indicates the category to be set; alternatively, LC_ALL is used to set all categories. The second argument is the locale name to be chosen, or alternatively empty string ("") is used to rely on system environment setting.

Therefore, the program initialization of a typical internationalized C program may appear as follows:

```
#include <locale.h>
...
const char *prev_locale;
prev_locale = setlocale (LC_ALL, "");
```

and the system environments are looked up to determine the appropriate locale as follows:

1. If LC_ALL is defined, it shall be used as the locale name.
2. Otherwise, if corresponding values of LC_CTYPE, LC_COLLATE, LC_MESSAGES are defined, they shall be used as locale names for corresponding categories.
3. For categories that are still undefined by the above checks, and LANG is defined, this is used as the locale name.
4. For categories that are still undefined by the above checks, "C" (or "POSIX") locale shall be used.

The "C" or "POSIX" locale is a dummy locale in which all behaviours are C defaults (e.g. ASCII sort for LC_COLLATE).

LC_CTYPE

LC_CTYPE defines character classification for functions declared in <*ctype.h*>:

- `iscntl()`
- `isgraph()`
- `isprint()`
- `isspace()`
- `ispunct()`
- `isalnum()`
- `isalpha()`
- `isdigit()`
- `isxdigit()`
- `islower()`
- `isupper()`
- `tolower()`
- `toupper()`

Since glibc is Unicode-based, and all character sets are defined as Unicode subsets, it makes no sense to redefine character properties in each locale. Typically, the LC_CTYPE category in most locale definitions refers to the default definition (called "i18n").

LC_COLLATE

C functions that are affected by LC_COLLATE are *strcoll()* and *strxfrm()*.

- *strcoll()* compares two strings in a similar manner as *strcmp()* but in a locale-dependent way. Note that the behaviour *strcmp()never* changes under different locales.
- *strxfrm()* translates string into a form that can be compared using the plain *strcmp()* to get the same result as when directly compared with *strcoll()*.

The LC_COLLATE specification is the most complicated of all locale categories. There is a separate standard for collating Unicode strings, called *ISO/IEC 14651 International String Ordering*.[29] The glibc default locale definition is based on this standard. Locale developers may consider investigating the *Common Tailorable Template (CTT)* defined there before beginning their own locale definition.

In the CTT, collation is done through multiple passes. Character weights are defined in multiple levels (four levels for ISO/IEC 14651). Some characters can be ignored (by using "IGNORE" as weight) at first passes and be brought into consideration in later passes for finer adjustment. Please see ISO/IEC 14651 document for more details.

LC_TIME

LC_TIME allows localization of date/time strings formatted by the *strftime()* function. Days of week and months can be translated into the locale language, appropriate date

LC_NUMERIC & LC_MONETARY

Each culture uses different conventions for writing numbers, namely, the decimal point, the thousand separator and grouping. This is covered by LC_NUMERIC.

LC_MONETARY defines currency symbols used in the locale as per ISO 4217, as well as the format in which monetary amounts are written. A single function *localeconv()* in <*locale.h*> is defined for retrieving information from both locale categories. Glibc provides an extra function *strfmon()* in <*monetary.h*> for formatting monetary amounts as per LC_MONETARY, but this is not standard C function.

LC_MESSAGES

LC_MESSAGES is mostly used for message translation purposes. The only use in POSIX locale is the description of a yes/no answer for the locale.

ISO/IEC 14652

The ISO/IEC 14652 *Specification method for cultural conventions*[30] is basically an extended POSIX locale specification. In addition to the details in each of the six categories, it introduces six more:

Category	**Description**
`LC_PAPER`	paper size
`LC_NAME`	personal name format
`LC_ADDRESS`	address format
`LC_TELEPHONE`	telephone number
`LC_MEASUREMENT`	measurement units
`LC_VERSION`	locale version

All of the above categories have already been supported by glibc. C applications can retrieve all locale information using the *nl_langinfo()* function.

Building Locales

To build a locale, a *locale definition* file describing data for ISO/IEC 14652 locale categories must be prepared. (See the standard document for the file format.) In addition, when defining a new character set, a *charmap* file must be created for it; this gives every character a symbolic name and describes encoded byte strings.

In general, glibc uses UCS symbolic names (<U*xxxx*>) in locale definition, for convenience in generating locale data for any charmap. The actual locale data to be used by C programs is in binary form. The locale definition must be compiled with the *localedef* command, which accepts arguments like this:

```
localedef [-f <charmap>] [-i <input>] <name>
```

For example, to build th_TH locale from locale definition file th_TH using TIS-620 charmap:

[29]ISO/IEC, ISO/IEC JTC1/SC22/WG20 – Internationalization; available from *anubis.dkuug.dk/jtc1/sc22/wg20*.

[30]Ibid.

```
# localedef -f TIS-620 -i th_TH th_TH
```

The charmap file may be installed at /usr/share/i18n/charmaps directory, and the locale definition file at /usr/share/i18n/locales directory, for further reference.

The *locale* command can be used with *"-a"* option to check for all installed locales and *"-m"* option to list supported charmaps. Issuing the command without argument shows the locale categories selected by environment setting.

Translation

The translation framework most commonly used in FOSS is *GNU gettext,* although some cross-platform FOSS, such as AbiWord, Mozilla and OpenOffice.org use their own frameworks as a result of the cross-platform abstractions. In this section, the GNU gettext, which covers more than 90 percent of GNU/Linux desktops, is discussed briefly. The concepts discussed here, however, apply to other frameworks.

Messages in program source code are put in a short macro that calls a gettext function to retrieve the translated version. At program initialization, the hashed message database corresponding to LC_MESSAGES locale category is loaded. Then, all messages covered by the macros are translated by quick lookup during program execution. Therefore, the task of translation is to build the message translation database for a particular language and get it installed in an appropriate place for the locale. With that preparation, the gettext programs are automatically translated as per locale setting without having to touch the source code.

GNU gettext also provides tools for creating the message database. Two kinds of files are involved in the process:

- *PO (Portability Object) file.* This is a file in human-readable form for the translators to work with. It is named so because of its plain-text nature, which makes it portable to other platforms.
- *MO (Machine Object) file.* This is a hashed database for machines to read. It is in the final format to be loaded by the gettext program. There are many translation frameworks in commercial Unices, and these MO files are not compatible. One may also find some GMO files as immediate output from GNU gettext tools. They are MO files containing some GNU gettext enhanced features.

Important GNU gettext tools will be discussed by describing the summarized steps of translation from scratch (See Figure 3):

1. Extract messages with the *xgettext* utility. What you get is the "*package*.pot" file as a template for the PO file.

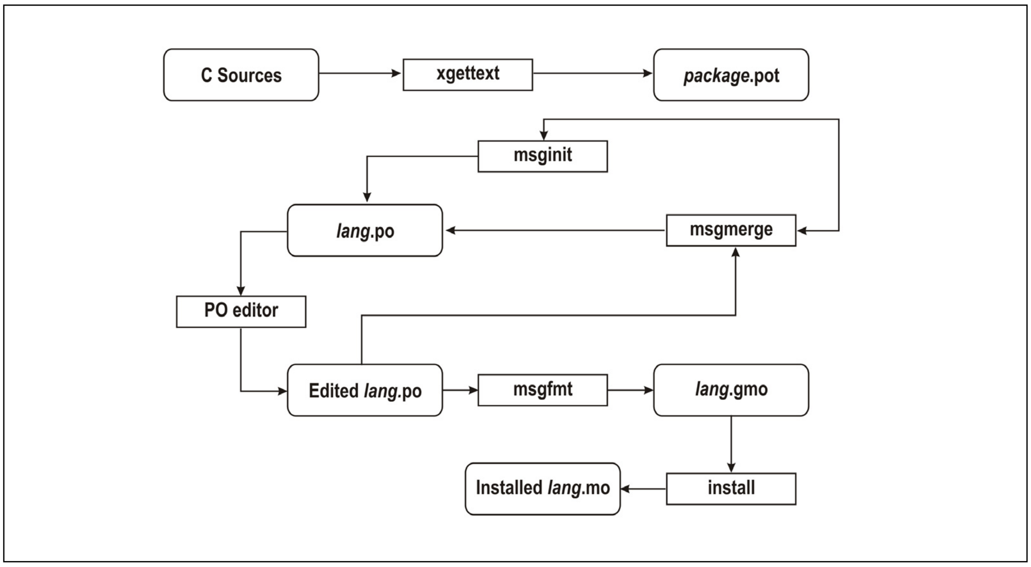


Figure 3 GNU gettext Working Process

2. Create the PO file for your language from the template, either by copying it to "*xx*.po" (where *xx* is your locale language) and filling its header information with your information, or by using the *msginit* utility.
3. Translate the messages by editing the PO file with your favourite text editor. Some specialized editors for PO files, such as kbabel and gtranslator, are also available.
4. Convert the PO file into MO file using the *msgfmt* utility.
5. Install the MO file under the LC_MESSAGES directory of your locale.
6. When the program develops, new strings are introduced. You need not begin from scratch again. Rather, you extract the new PO template with the *xgettext* utility as usual, and then merge the template with your current PO with the *msgmerge* utility. Then, you can continue by translating the new messages.

GNOME intltool

GNU/Linux desktops have more things to translate than messages in C/C++ source code. The system menu entries, lists of sounds on events, for example, also contain messages, mostly in XML formats that are not supported by GNU gettext. One may dig into these individual files to translate the messages, but this is very inconvenient to maintain and is also error prone.

KDE has a strong policy for translation. PO files for all KDE core applications are extracted into a single directory for each language, so that translators can work in a single place to translate the desktop without a copy of the source code. But in practice, one needs to look into the sources occasionally to verify the exact meaning of some messages, especially error messages. This already includes all the messages outside the C++ sources mentioned above.

GNOME comes up with a different approach. The PO files are still placed in the source under the "po" subdirectory as usual. But instead of directly using *xgettext* to extract messages from the source, the GNOME project has developed an automatic tool called *intltool*. This tool extracts messages from the XML files into the PO template along with the usual things *xgettext* does, and merges the translations back as well. As a result, despite the heterogeneous translation system, what translators need to do is still edit a single PO file for a particular language.

The use of *intltool* is easy. To generate a PO template, change the directory to the "po" subdirectory and run:

```
$ intltool-update --pot
```

To generate a new PO file and merge with existing translation:

```
$ intltool-update xx
```

where *xx* is the language code. That is all that is required. Editing the PO file as usual can then begin.

When PO editing is complete, the usual installation process of typical GNOME sources will automatically call the appropriate *intltool* command to merge the translations back into those XML files before installing. Note that, with this automated system, one should not directly call the *xgettext* and *msgmerge* commands any more.

The following sites and documents provide more information on KDE and GNOME translation:

- KDE Internationalization Home (*i18n.kde.org/*)
  - The KDE Translation HOWTO (*i18n.kde.org/translation-howto/*)
- The GNOME Translation Project (*developer.gnome.org/projects/gtp/*)
  - Localizing GNOME Applications (*developer.gnome.org/projects/gtp/l10n-guide/*)
  - How to Use GNOME CVS as a Translator (*developer.gnome.org/doc/tutorials/gnome-i18n/translator.html*)

PO Editors

A PO file is a plain text file. This can be edited, using a favourite text editor. But, as stated earlier, translation is a labour-intensive task. It is worth considering some convenient tools to speed up the job.

Normally, the editor is needed to be able to edit UTF-8, as both KDE and GNOME now have used it as standard text encoding. However, the following tools have many other features.

KBabel

Part of the KDE Software Development Kit, KBabel is an advanced and easy-to-use PO-files editor with full navigation and editing capabilities, syntax checking and statistics. The editor separates translated, un-translated and fuzzy messages so that it is easy to find and edit the unfinished parts.

KBabel also provides CatalogManager, which allows keeping track of many PO-files at once, and KBabelDict for keeping the glossary, which is important for translation consistency, especially among team members from different backgrounds.

Gtranslator

Gtranslator is the PO-file editor for the GNOME desktop. It is very similar to Kbabel in core functionality.

Gtranslator also supports auto-translation, where translations are learnt and transferred into its memory, and can be applied in later translations using a hot key.

FURTHER READINGS

Raymond, 2001

Raymond, Eric S., *The Cathedral & the Bazaar*, O'Reilly, 2001.

Named after the classic article that has motivated the Open Source movement, this book presents the article along with other opinions of the same author. The article analyzes how the GNU/Linux kernel has been developed with such dramatic speed as compared to proprietary projects, despite being a "hobby" project done by thousands of contributors around the world. It is interesting to note how a project of that size is managed and how this phenomenon occurs. Some principles are extracted for use in general software projects, exemplified by the *fetchmail* project done by the author. The term "Open Source" was then coined as a proposed replacement to the "Free Software" term used by former campaigns led by the GNU project, as it was less ambiguous and more friendly to businesses.

The article is available at *catb.org/~esr/writings/cathedral-bazaar/*.

DiBona , Ockman & Stone, 1999

DiBona, C., Ockman, S. and Stone M., eds., *Open Sources – Voices from the Open Source Revolution,* O'Reilly, 1999.

This includes articles by leaders of the FOSS movements: Brian Behlendrof (Apache), Kirk McKusick (Berkeley Unix), Tim O'Reilly (O'Reilly & Associates), Bruce Perens (Debian, Open Source Initiative), Tom Paquin and Jim Hamerly (mozilla.org, Netscape), Eric Raymond (Open Source Initiative), Richard Stallman (GNU, Free Software Foundation, Emacs), Michael Tiemann (Cygnus Solutions), Linus Torvalds (Linux), Paul Vixie (Bind), Larry Wall (Perl). The strategy and success of FOSS is documented in these articles.

Williams, 2002

Williams, S., *Free as in Freedom – Richard Stallman's Crusade for Free Software*, O'Reilly, 2002.

This is the story of Richard Stallman, the founder of the GNU project, and the birth of the Free Software movement. It examines Stallman's unique personality and how it has been both a driving force in terms of the movement's overall success.

Unicode

The Unicode Consortium, *The Unicode Standard, Version 4.0.1, defined by: The Unicode Standard, Version 4.0*, Addison-Wesley, Reading, MA, 2003. As amended by *Unicode 4.0.1*. (*www.unicode.org/versions/ Unicode4.0.1/*).

This explains the Unicode standard, including descriptions of planes, code charts and implementation guidelines.

ISO10646

ISO/IEC 10646:2003, *Information Technology – Universal multi-octet character set – UCS*, 2003; available from *std.dkuug.dk/JTC1/SC2/WG2/*.

The ISO/IEC 10646 UCS standard is explained, with code chart and conformance specifications.

Kuhn

Kuhn, M., *UTF-8 and Unicode FAQ for Unix/Linux;* available from *www.cl.cam.ac.uk/~mgk25/unicode.html.*

A comprehensive one-stop information resource on how to use Unicode/UTF-8 on POSIX systems as well as the current status of support.

Graham, 2000

Graham, T., *Unicode – Primer*, IDG Books Worldwide, 2000.

ISO14652

ISO/IEC TR 14652:2002(E), *Information Technology – Specification method for cultural conventions*, 2002.

The specification for creating locale definitions on which GNU C library is based.

ISO14651

ISO/IEC DIS 14651, *International string ordering and comparison – Method for comparing character strings and description of the common template tailorable ordering,* 2000.

This is a basic template for UCS string collation, from which LC_COLLATE category in locale definitions can be modified to fit local cultures.

Adobe & Microsoft

Adobe, Microsoft, *OpenType Specification v.1.4.*; available from *partners.adobe.com/asn/developer/opentype/main.html* and *www.microsoft.com/typography/otspec/default.htm.*

Adobe

Adobe, *Adobe Solutions Network – Type Technology*, available from *partners.adobe.com/asn/tech/type/index.jsp.*

Specifications and conventions for font technology related to Adobe Postscript®. A must for standard-conformant font creations.

Microsoft

Microsoft, *Microsoft Typography – Specifications*; available from *www.microsoft.com/typography/specs/default.htm.*

Specifications for OpenType and TrueType font technologies. Also included are specifications for fonts of specific scripts and languages as required by Microsoft Uniscribe, the Unicode script processor.

William

William, G., *FontForge – An Outline Font Editor*; available from *fontforge.sourceforge.net/overview.html.*

A tutorial for FontForge. One can learn from it even when using other font editors.

Gettys & Scheifler, 1996

Gettys, J. and Scheifler, R.W., *Xlib – C Language X Interface*, X Consortium Standard, X Version 11 Release 6.4, 1996.

The reference for X11R6.4 architecture that you can consult when implementing at X Window level.

Nye, 1995

Nye, A., ed., *Programmer's Supplement for Release 6 of the X Window System,* O'Reilly, 1995.

A guide to the features introduced in X Window Version 11 Release 6 (X11R6). The internationalization chapter may be the most interesting for readers.

Packard, 2002

Packard, K., *Font Configuration and Customization for Open Source Systems.* GUADEC 2002, Seville, 2002; available from *keithp.com/~keithp/talks/guadec2002/.*

Documents the background of the FontConfig library during the initial phases.

Palmer, 2003

Palmer, D., *An Unreliable Guide to XKB Configuration*, 2003; available from *www.charvolant.org/~doug/xkb/.*

A document on XKB written by a hacker frustrated by the lack of documentation.

Pascal

Pascal, I., *How to Configure XKB*; available from *www.tsu.ru/~pascal/en/xkb.*

A complete reference for XKB configuration.

Hiura, 1999

Hiura, H., *Internet/Intranet Input Method Architecture.* (*www.openi18n.org/subgroups/im/IIIMF/whitepaper/whitepaper.html*).

A white paper describing the concepts of IIIM, the next generation cross-platform input method architecture which should soon replace XIM.

Tuoc, Lok & Taylor, 1995

Tuoc L V., Lok J.S.H. and Taylor D.J., *Internationalization: Developing Software for Global Markets*, John Wiley & Sons, 1995.

Sprung, 2000

Sprung, R., ed., *Translating Into Success: Case Studies in Translation, Localization, and Internationalization*, John Benjamins, 2000.

Lunde, 1999

Lunde, K., *CJKV Information Processing*, O'Reilly,1999; available from *www.oreilly.com/catalog/cjkvinfo.*

Ott, 1999

Ott, C., *Global Solutions for Multilingual Applications*, Wiley, 1999.

RESOURCES AND TOOLS

Bangladesh

www.bengalinux.org
Ankur – Supporting Bangla (Bengali) on GNU/Linux.

biosindex.blogspot.com/
Bangla Innovation Through Open Source.

www.bdlug.org
Bangladesh Linux User Group.

Cambodia

www.bauhahnm.clara.net/Khmer/Welcome.html
Resources on Khmer font in Unicode.

www.khmercivil.com/unicode/index.php
Khmer Unicode Windows.

www.khmeros.info
Khmer Open Source.

China

www.linux.org.cn
Linux China.

www.gnuchina.org
GNU China.

www.redflag-linux.com/eindex.html
Red Flag Linux.

www.linuxforum.net
China Linux Forum.

www.linuxeden.com
Free Software.

linux.softhouse.com.cn
Linux Home.

www.cosix.com.cn
China software.

www.xteam.com.cn
Xteam Software.

www.turbolinux.com.cn
TurboLinux China.

India

www.linux-india.org
Linux India.

www.gnu.org.in
Free Software Foundation of India.

rohini.ncst.ernet.in/indix
Linux Localization Team, National Centre for Software Technology.

hi.openoffice.org
OpenOffice localization project for Hindi.

www.indlinux.org
Indian localization projects.

Indonesia

www.itb.ac.id
Indonesian Open Source Contributor Group.

www.pandu.org
Indonesian Open Source Community.

opensource-indonesia.com/kioss.php/index.php
Open Source Indonesia.

gnome.linux.or.id
Gnome Indonesia.

Lao PDR

sabaidi.imag.fr/laounicode-en.htm
Lao Unicode and related resources.

park.lao.net/references
Various documents on Lao: fonts, encoding schemes, and locales.

laonux.muanglao.com
Lao KDE localization project.

xangdao.muanglao.com
OpenOffice Lao version project.

Myanmar

www.myanmarlug.org/pegunc-linux
Linux Project.

www.myanmarlinux.org
Myanmar Linux User Groups.

sourceforge.net/projects/mandalay
Myanmar Language Support Project.

Malaysia

www.my-opensource.org
Malaysia Open Source.

kde-ms.sourceforge.net
KDE Malaysia.

oo-l10n-my.sourceforge.net
OpenOffice Malaysia.

Mongolia

openmn.sourceforge.net
Open Source Initiative.

Nepal

www.linux.com.np
Nepal Linux User Group.

www.linuxnepal.com.np
Nepal Linux.

Pakistan

www.linuxpakistan.net
Linux Pakistan.

Philippines

plug.linux.org.ph
Philippines Linux User Group.

opensource.asti.dost.gov.ph
Open Source Foundation.

www.bayanihan.gov.ph
Linux Localization Project.

www.asti.dost.gov.ph
Government Open Source Initiatives.

osl.apc.edu.ph
Open Source Centre.

Sri Lanka

www.lklug.pdn.ac.lk/index.php
Lanka Linux User Group.

www.opensource.lk
Lanka Open Source Foundation.

www.tamil.homelinux.org
Tamil Linux.

Thailand

linux.thai.net
Thai Linux Working Group.

opensource.thai.net
Thai Open Source Network.

opentle.org
Linux TLE, OfficeTLE.

www.pladao.org
Pladao Office.

www.nectec.or.th/it-standards
Thailand's IT standards.

www.tosf.org
Thai Open Source Federation.

tosdn.com
Thai Open Source portal.

th.openoffice.org
OpenOffice Thai Project.

Viet Nam

www.linuxvn.com
Linux Viet Nam.

www.linux.cmc.com.vn
Linux Forum.

h0lug.sourceforge.net
Ho Chi Minh Linux User Group.

vi.i18n.kde.org
Vietnamese KDE Localization Project.

gnomevi.sourceforge.net
Vietnamese Gnome Localization Project.

www.asiaosc.org/uploads/20030313_1/downloads/TranLuuChuong/Slide/html/img0.html
Chuong T.L., Ministry of Science & Technology, Minh H.T., CMC Co Ltd.

www.ssc.com:8080/glue/groups/nonus/vietnam
Linux User Groups in Viet Nam.

www.isoc-vn.org/www/index-vn.html
ISOC - Internet Society Viet Nam.

linux.vietkey.net
Linux Viet Nam - VietKey Group.

Localization Information Portals and Tools

www.microsoft.com/globaldev
Microsoft's global development portal.

www.translation.net
A portal with information, software and links on translation.

www.i18ngurus.com and *www.i18n.com*
News, tools, processes for global software

www.kde.org and *www.gnome.org*
KDE and GNOME

www.openoffice.org
OpenOffice.org

www.mozilla.org
Mozilla.

Organizations

www.unicode.org
The official Web site of the Unicode Consortium.

www.hclrss.demon.co.uk/unicode
A Unicode resources website with links to variety of useful tools and utilities e.g. Unicode fonts, shareware, etc.

www.czyborra.com/unicode/characters.html
An introduction to Unicode with many useful links.

ftp://dkuug.dk/i18n/index.htm
Internationalization standards and tools.

www.lisa.org
Localization Industry Standards Association (LISA).

www.opentag.com
This site is dedicated to the tools and technologies used in the localization of software, online help and documentation.

www.iso.ch
The International Standards Organization (ISO).

GLOSSARY

ASCII

American Standard Code for Information Interchange.

Character

Logical unit that represents a letter, digit or symbol of a writing system.

Character Set

The mapping of characters from a writing system to a set of binary codes.

CJK

Chinese, Japanese and Korean. Usually used to refer to the common Han ideographs shared by th three languages and other issues surrounding this. Sometimes associated with Vietnamese and called CJKV.

DNS

Domain Name Server, a service that resolves symbolic host names into numeric IP addresses, and vice versa.

Encoding

A character encoding scheme is a set of rules for representing a sequence of character codes with byte sequence.

Font

Resource that provide a set of glyphs for text rendering.

Fontconfig

A client-side font configuration library split from Xft as an X-independent module. Modern GNU/Linux desktops are using it as default font system, as opposed to the traditional font system managed by X server.

FOSS

Free/Open Source Software.

Glyph

Graphical unit for composing text display of a writing system. A glyph may represent the shape of a single character, part of a character (in the case of characters with multiple parts), or a combined form of character sequence (in the case of ligatures).

GNOME

GNU Network Object Modelling Environment, a desktop environment based on GTK+ toolkit and other desktop components.

GNU

A recursive acronym standing for "GNU's Not Unix." GNU is an effort to create a completely free operating system based on Unix architecture.

GTK+

The GIMP Tool Kit, a toolkit initially created for using with The GIMP (GNU Image Manipulation Program), but later becomes an independent general-purpose toolkit. GTK+ is written in C, licensed under LGPL.

I18N

Abbreviation for Internationalization.

IIIMF

Internet/Intranet Input Method Framework, a new framework for cross-platform input method developed by OpenI18N.org. IIIMF bridges different IM protocols by using wrappers that communicate with a common protocol.

Internationalization

The practice of developing software with the readiness to work in different languages and cultures without modification of the source code or re-compilation. This usually means abstraction in culture-sensitive operations as well as defining the mechanism for dynamically loading locale-based implementations.

KDE

K Desktop Environment, a desktop environment based on Qt toolkit and other desktop components.

Kernel

A very low-level software that manages computer hardware, multi-tasks the many programs that are running at any given time, and other such essential things.

L10N

Abbreviation for Localization.

Locale

The features of the user's environment that are dependent on language, country and cultural conventions. The locale determines conventions such as string collation; date and time formats; number and currency formats; and so on.

Localization

Implementation of cultural conventions defined by the internationalization process according to different languages and cultures.

Multilingual

Supporting more than one language simultaneously. Often implies the ability to handle more than one script and character set.

OTF

OpenType Font.

Pango

A Unicode-based multi-lingual text rendering engine used by GTK+ 2. Like GTK+, Pango is written in C and licensed under LGPL.

PHP

A server-side scripting language for creating dynamic web pages.

POSIX

Portable Operating System Interface Specification is the minimum specification of system calls for operating systems based on Unix, defined by IEEE so that applications based on it are guaranteed to be portable across OSs. Although based on Unix, POSIX is also supported by some non-Unix OSs.

Qt

A cross-platform toolkit developed by TrollTech, and later used as the base for the KDE project. Qt is written in C++ and dual-licensed under GPL and QPL.

Script

A system of characters used to write one or several languages.

SSH

Secure Shell is used for remote login using an encrypted connection to prevent sniffing by third parties.

Toolkit

A library for GUI application development. It provides common widgets and controls such as menu, button, text entry, etc.

TTF

TrueType Font.

UCS

Universal Multi-octet coded character set, as defined by ISO/IEC 10646 to represent the world's writing systems. It is maintained by ISO/IEC JTC1/SC2/WG2, with contributions from the Unicode Consortium.

Unicode

A 24-bit coded character set used to represent the world's writing systems. It is maintained by the Unicode Consortium, in synchronization with ISO/IEC 10646.

UTF-8

Unicode (UCS) Transformation Format, using 8-bit multibyte encoding scheme.

X Client

Application process that makes requests to the X server for displaying graphical outputs on the screen and/or accepts events from input device. (*See also:* X Server.)

X Font Server

A service that provides font accesses to the X server.

XFS

X Font Server.

Xft

An extension introduced in XFree86 for managing glyph images of vector fonts at client side. This allows X applications to add special effects to the glyphs, such as anti-aliasing.

XFree86

A free X Window implementation for PC, although Mac is also supported. It is developed on top of the free source code from the X consortium (X.org), along with hardware drivers and some extensions. It used to be the only X Window implementation for FOSS on PC, until recent forks[1] by efforts such as Xouvert, freedesktop.org X server, Keith Packard's kdrive, and X11R6.7, the latest one which is endorsed by X.org itself. (*See also:* X.org.)

XIM

X Input Method framework, introduced in X11R6 as part of its I18N. It makes complicated text input processes, such as those for CJK, possible.

XKB

X Keyboard extension, introduced in X11R6 as part of its I18N. It provides a method to describe keyboard layouts, to combine multiple layouts, and to switch between them.

XOM

X Output Method framework, introduced in X11R6 but fully implemented in X11R6.5.1 to provide complex text rendering. But its late arrival, when rendering engines like Pango have already matured, makes its usefulness questionable.

X.org

The X consortium that maintains the X Window system. Normally, it maintains the X specification and provides the framework implementation source code in free license. In early 2004, triggered by license change in XFree86, a group of developers forked XFree86 source and made it X11R6.7 release in the name of X.org.

X Server

The service that provides interface to graphical hardware and input devices in the X Window system. Its clients are called *X Clients.* (*See also:* X Client.)

X Window

A graphical environment initially developed by the Athena project at MIT with support from some vendors, and later maintained by the X consortium. X Window is the major graphical environment for most Unix variants nowadays.

[1]In software, a project fork or branch happens when a developer (or a group of them) takes code from a project and starts to develop independently of the rest. This schism can be caused because of different goals or personality clashes. Definition available at *www.wordiq.com/definition/Fork_(software)*

ABOUT THE AUTHORS

Anousak Souphavanh is a computer scientist, an ICT consultant and a GNU/Linux enthusiast. He obtained his B.S. in Computer Science from Binghamton University's Watson School of Engineering and Applied Science, The State University of New York and his Masters in Computer Science from the Rochester Institute of Technology, New York. He was a senior systems administrator, technical lead person and network specialist for IBM in Rochester from 1995 to 2003. In 2003, he was an IT project manager for Jhai Foundation, managing a rural area IT project in Lao PDR. He is currently an IT consultant. He is an author of Laonux (*Laonux.muanglao.com*), the Lao version of GNU/Linux (KDE). He is also working on several other projects for Lao language: OpenOffice.org, PHPNuke, and Mozilla. He is an active member of the Lao Open Source community. His interest in GNU/Linux led him to initiate the Lao Linux User Group in Vientiane, Laos in 2003. He developed the Lao locale definition for GNU C library. He is also a leading voice on Lao support in GNU/Linux desktops with several working groups.

Theppitak Karoonboonyanan is a member of the Thai Linux Working Group (TLWG). He obtained his B.Eng. (Honors) in Computer Engineering from Chulalongkorn University, Thailand, in 1993.

He has contributed significantly to Thai GNU/Linux localization. His first contribution was an enhanced version of Thai string collation rules based on the Royal Institute's model. He then developed the Thai locale definition for GNU C library based on it. He has contributed to many FOSS projects to establish Thai support infrastructure for GNU/Linux desktop, including XFree86, Pango, GLib, and GTK+.

As a TLWG member, he helped the LibThai project to collect and develop routines for basic Thai Language processing, and currently maintains the Thai LaTeX project and contributes to *thaifonts-scalable*, a Thai font development project. Also, he tracks the overall status of Thai support in GNU/Linux desktops and occasionally updates the group to keep up the momentum of development.

He was a member of Linux TLE and was among the founding members of GNU/Linux localization project at the National Electronics and Computer Technology Center (NECTEC). Currently, he freelances and has contributed advice and experience to the development of Laonux, a Lao KDE-based GNU/Linux desktop, and developed solutions for other clients.

APDIP

The Asia-Pacific Development Information Programme (APDIP) is an initiative of the United Nations Development Programme (UNDP) that aims to promote the development and application of new Information and Communication Technologies (ICTs) for poverty alleviation and sustainable human development in the Asia-Pacific region. It does so through three core programme areas, namely, Policy Development and Dialogue; Access; and Content Development and Knowledge Management.

In collaboration with national governments, APDIP seeks to assist national and regional institutions in the Asia-Pacific through activities that involve awareness-raising and advocacy, building capacities,promoting ICT policies and dialogue, promoting equitable access to tools and technologies, knowledge sharing, and networking. Strategic public-private sector partnerships and opportunities for technical cooperation among developing countries (TCDC) are APDIP's key building blocks in implementing each programme activity.

www.apdip.net

IOSN

The International Open Source Network (IOSN) is an initiative of UNDP's Asia-Pacific Development Information Programme (APDIP). Its overall objective is to serve as a Center of Excellence and a Clearinghouse for Information on Free and Open Source Software (FOSS) in the Asia-Pacific region. In order to accelerate the world-wide adoption of FOSS, IOSN seeks to raise awareness, facilitate networking of people, strengthen capacities, and conduct R&D.

The beneficiaries of IOSN are governments, IT professionals, software developers, the FOSS R&D community, academics and the NGO community. IOSN serves as a resource center to help policymakers and decision-makers in the public sector, educational institutions, businesses and others develop policies and plans for the use of FOSS in their respective organizations. Much of IOSN's activities are undertaken online and the IOSN portal has been developed for this purpose and serves as a comprehensive online resource center on FOSS. The IOSN portal also provides a means for the FOSS community in the region to contribute to its efforts and to interact with each other.

www.iosn.net