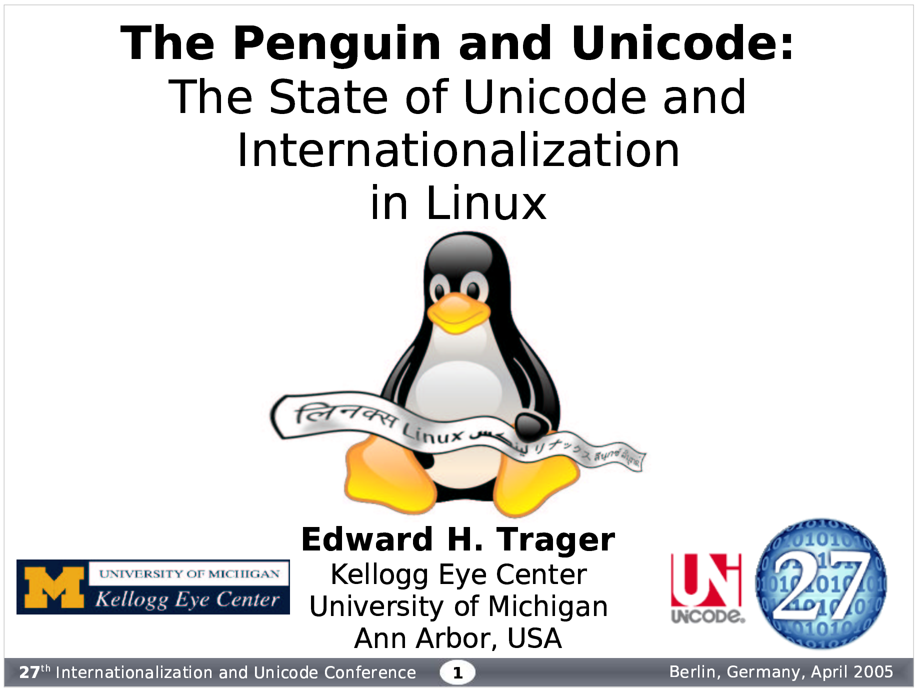

लिनक्स Linux لينكس リナックス

The Penguin and Unicode:
The State of Unicode and Internationalization in Linux

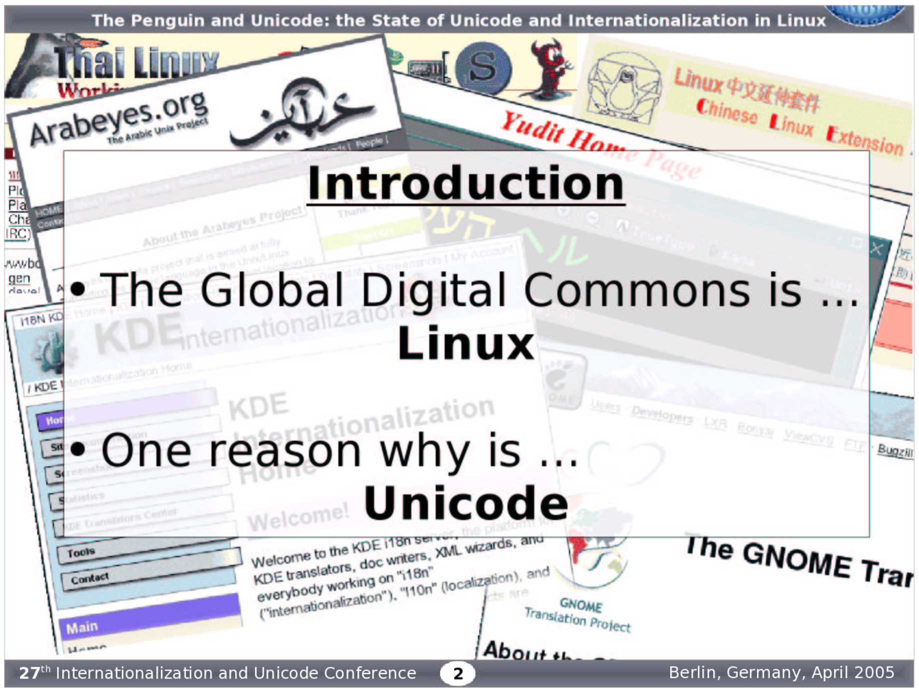
Edward H. Trager
Kellogg Eye Center
University of Michigan
Ann Arbor, USA

27th Internationalization and Unicode Conference 1 Berlin, Germany, April 2005

The Penguin and Unicode:
The state of Unicode and Internationalization in Linux

Presented by Edward H. Trager
Kellogg Eye Center,
University of Michigan, Ann Arbor

27th Internationalization and Unicode Conference,
Berlin, Germany, April 8, 2005



The Penguin and Unicode: the State of Unicode and Internationalization in Linux

Introduction

- The Global Digital Commons is … **Linux**
- One reason why is … **Unicode**

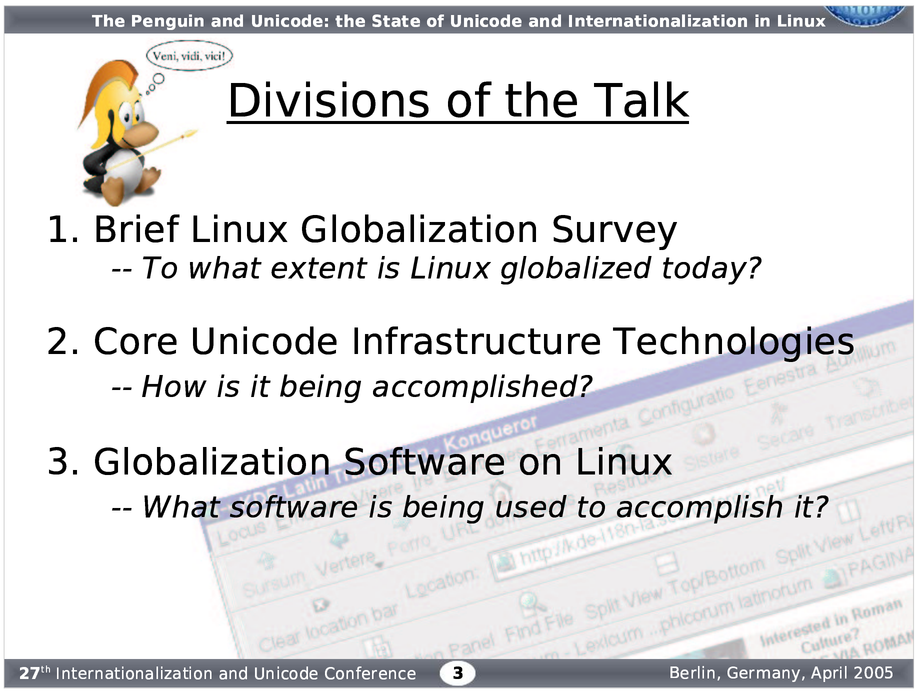
27th Internationalization and Unicode Conference 2 Berlin, Germany, April 2005

Introduction: The Open Source Digital Commons is Linux

Linux –and not just Linux on the server, but Linux on the desktop– is now a global phenomenon of massive proportions. Why is that? Of course, there are many reasons. In many countries, the lower cost of Linux coupled with the ability of the operating system to run well on older hardware are certainly key factors contributing to widespread adoption of this Open Source operating system. However another key reason is … Unicode.

The open and free nature of Linux technologies is spurring a rapid global movement to make Linux fully accessible in numerous languages and emerging markets. Businesses and organizations who understand that Linux is becoming the backbone of a global *Digital Commons* stand poised to benefit from the freedoms, technical opportunities, and cost savings of this open source platform.

In this talk, I will discuss how Unicode is being implemented in key Linux technologies from "glibc" to "X". Although this is a technical talk, I intend to provide a broad survey that I hope will allow you to “see the forest for the trees” and thus enable you –regardless of whether you are a developer, manager, or just a curious user– to make informed decisions about Linux internationalization technologies and globalization software.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

Divisions of the Talk

Veni, vidi, vici!

1. Brief Linux Globalization Survey
   -- *To what extent is Linux globalized today?*
2. Core Unicode Infrastructure Technologies
   -- *How is it being accomplished?*
3. Globalization Software on Linux
   -- *What software is being used to accomplish it?*

27th Internationalization and Unicode Conference 3 Berlin, Germany, April 2005

The talk is divided into three parts:

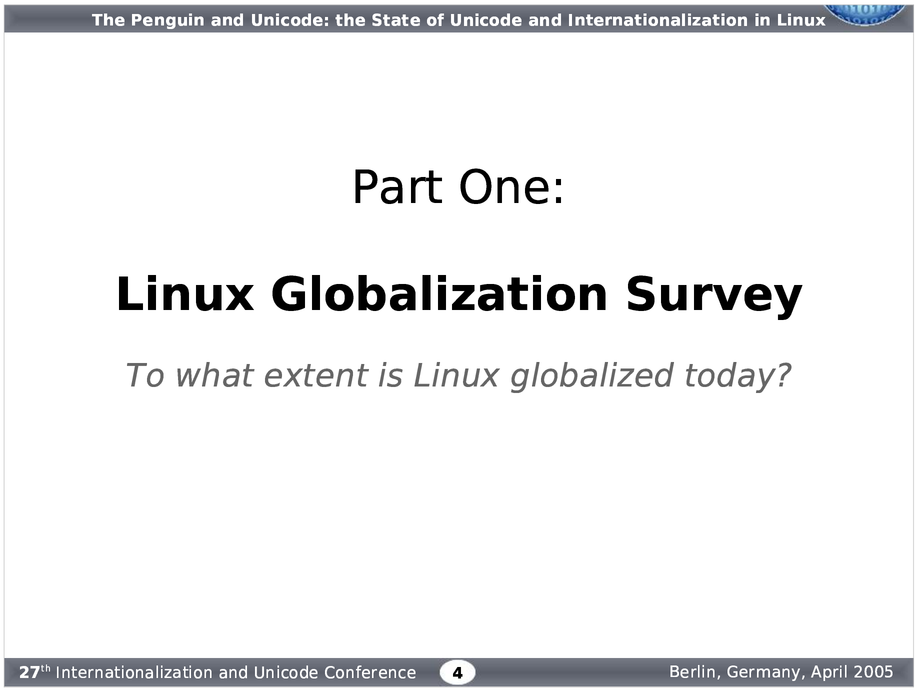
1. A brief Linux globalization survey.
One day while surfing the web I came across a project to localize the KDE desktop into *Latin*. This piqued my curiosity. Who would need their computer desktop in Latin? The Pope? Maybe. Classicists? Maybe. I was surprised to think that there are people out there who, instead of studying Cicero and Ovid, are trying to translate KDE into Latin. Perhaps they are following in the footsteps of Caesar: Veni, vidi, vici[1]! If KDE was being translated even into a classical language like Latin, I wondered into what other languages KDE and Gnome were being translated?

So, to what extent is Linux globalized today? To give you a feel of how big this phenomenon is, we will start off with an overview of the globalization of Linux.

2. A discussion of core Unicode infrastructure technologies in Linux.
Linux localization projects don't happen by magic. Besides linguistic, editorial and other skills, people need software tools to make all of this happen. And those software tools need to work on top of a solid, Unicode-based infrastructure. What is that infrastructure and how does it work? We'll take a quick tour in this part of the talk.

3. An introduction to software used in the globalization of Linux.
Finally, what are some of the tools that are facilitating projects like the KDE Latin localization and the rapid globalization of Linux in general? We'll explore a few of them in the final part of the talk.

1. "I came, I saw, I conquered!" Julius Caesar's report to the Senate after defeating Pharnaces II, king of Pontus in Asia Minor, in 47 B.C.E.



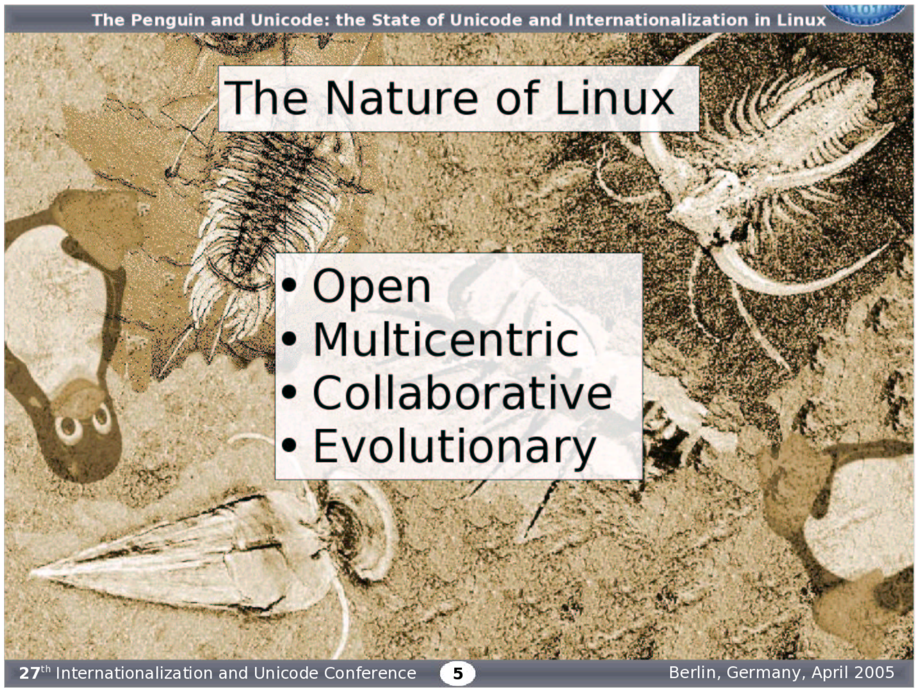
Part One:

Linux Globalization Survey

To what extent is Linux globalized today?

Part One: Linux Globalization Survey

In this part of the talk we'll explore the extent to which Linux is currently globalized …


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

The Nature of Linux

- Open
- Multicentric
- Collaborative
- Evolutionary

27th Internationalization and Unicode Conference 5 Berlin, Germany, April 2005

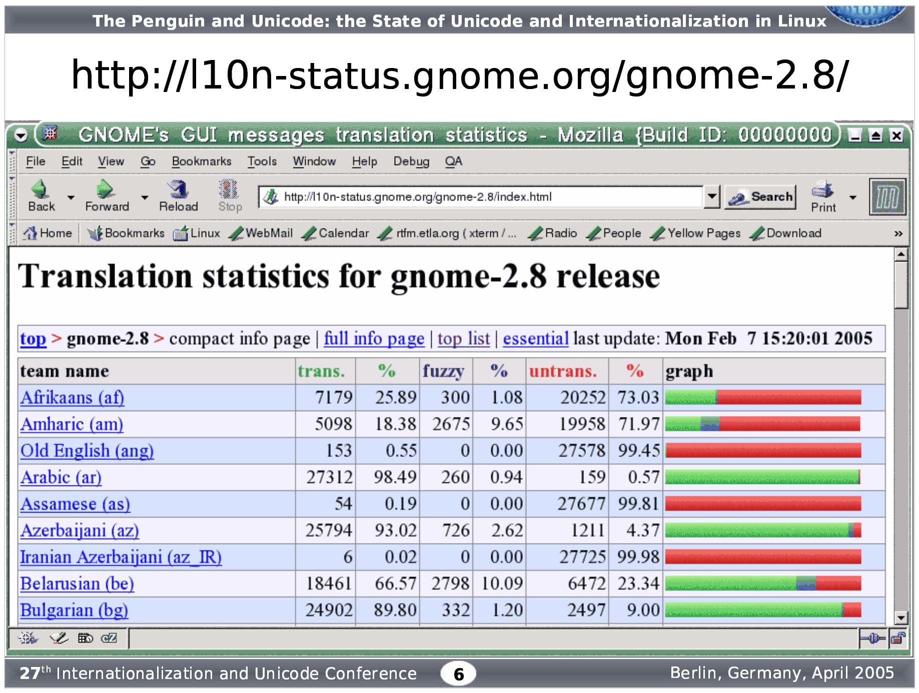
To set the stage for this talk, I'd like to review some aspects of the Open Source development process that impact Linux and the globalization of Linux in significant ways.

First, Linux is an *open* system. The “guts” of Linux are completely exposed for everyone to download from the internet and review. Not only is the source code available, but all of the data and meta data required for a complete system are also just a few downloads away. Locale data for hundreds of countries and regions are available. Localized translations for major components of the KDE and Gnome desktops, as well as for software like Mozilla/Firefox, are also freely available.

Second, Linux development is *multicentric* and *collaborative*. In addition to the big corporate players like Redhat, Novell SuSE, and IBM, we are now also witnessing the creation of non-profit foundations like the Mozilla Foundation[1] and the Apache Software Foundation[2]. These foundations are beginning to provide long-term vision, legal and financial security for Open Source projects that provide immense community value in their own right. Within the legal frameworks provided by Open Source licenses like the GNU General Public License (GPL)[3], everybody in the community can, and does, rely on the resources provided by others. In this multicentric, collaborative universe, similar but different projects, like the KDE and Gnome desktop projects, can co-exist and flourish.

Third, Linux development is *evolutionary*. With an extremely active and large world-wide community of developers, software evolves on very short time scales. In this system, software that is perceived by the community as having value usually evolves very rapidly to become extremely stable and functional. As people tend to enjoy things written in their own languages and using their own regional conventions, it is only natural that the localization of Linux is assigned a high priority, and thus it is proceeding rapidly.

1. Mozilla Foundation: http://www.mozilla.org.
2. Apache Software Foundation: http://www.apache.org.
3. GNU General Public License: http://www.gnu.org/copyleft/gpl.html.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

http://l10n-status.gnome.org/gnome-2.8/

GNOME's GUI messages translation statistics - Mozilla {Build ID: 00000000}
File Edit View Go Bookmarks Tools Window Help Debug QA
Back Forward Reload Stop
http://l10n-status.gnome.org/gnome-2.8/index.html
Search Print
Home Bookmarks Linux WebMail Calendar rtfm.etla.org (xterm / ... Radio People Yellow Pages Download

Translation statistics for gnome-2.8 release

top > **gnome-2.8** > compact info page | full info page | top list | essential last update: **Mon Feb 7 15:20:01 2005**

team name	trans.	%	fuzzy	%	untrans.	%	graph
Afrikaans (af)	7179	25.89	300	1.08	20252	73.03	🟩🟥
Amharic (am)	5098	18.38	2675	9.65	19958	71.97	🟩🟦🟥
Old English (ang)	153	0.55	0	0.00	27578	99.45	🟥
Arabic (ar)	27312	98.49	260	0.94	159	0.57	🟩
Assamese (as)	54	0.19	0	0.00	27677	99.81	🟥
Azerbaijani (az)	25794	93.02	726	2.62	1211	4.37	🟩🟦🟥
Iranian Azerbaijani (az_IR)	6	0.02	0	0.00	27725	99.98	🟥
Belarusian (be)	18461	66.57	2798	10.09	6472	23.34	🟩🟦🟥
Bulgarian (bg)	24902	89.80	332	1.20	2497	9.00	🟩🟥

27th Internationalization and Unicode Conference 6 Berlin, Germany, April 2005

Let's start by looking at the state of localization in the two most popular Linux desktops: KDE and Gnome. KDE has been around longer and is the more mature desktop, but Gnome has a very strong and enthusiastic developer base.

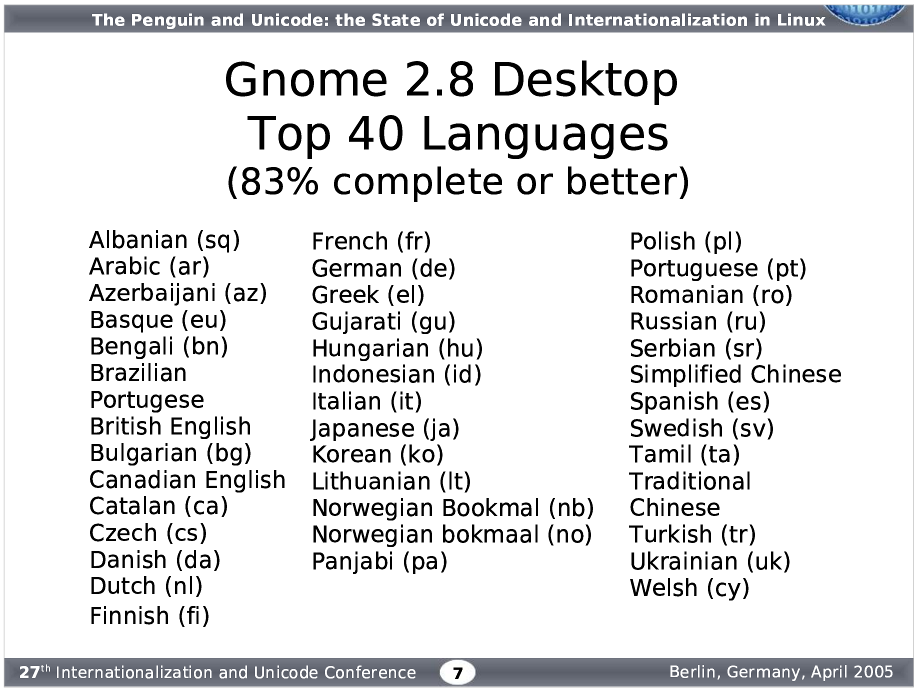
Gnome Desktop Localization

At the time of this writing (Feb. '05), the Gnome desktop project had 87 language localization projects listed on http://l10n-status.gnome.org. The completion status of each project is shown graphically on the site: green bars indicate the percentage of translated messages, red untranslated, and blue indicates "fuzzy".

Some of the more unusual languages represented on the site are Old English and Interlingua. Old English was used for a period of about 700 years, from the time of the Anglo-Saxon migrations into England around 450 A.D. until some time after the Norman Invasion in 1066. As far as I know, no one today speaks Old English. No matter, this project may find great favor with Medievalists. Currently, the project has only translated about 153 out of a total of more than 27,000 messages present in Gnome version 2.8.

Interlingua is a "planned auxilliary language" created by the International Auxilliary Language Association (IALA) in 1951. The Old English and Interlingua localization projects may be nothing more than interesting artifacts that are bound to turn up in an Open Source "ecosystem". On the other hand, if the Old English localization project survives, it might give new meaning to the term "Beowulf cluster"[1].

1. "Beowulf clusters" are scalable computer clusters based on commodity hardware using Open Source (Linux) software: see http://www.beowulf.org. "Beowulf" is also the title of the epic poem about a great Scandinavian warrior written in Old English sometime before the 10th century A.D.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

Gnome 2.8 Desktop Top 40 Languages (83% complete or better)

Albanian (sq)	French (fr)	Polish (pl)
Arabic (ar)	German (de)	Portuguese (pt)
Azerbaijani (az)	Greek (el)	Romanian (ro)
Basque (eu)	Gujarati (gu)	Russian (ru)
Bengali (bn)	Hungarian (hu)	Serbian (sr)
Brazilian Portugese	Indonesian (id)	Simplified Chinese
British English	Italian (it)	Spanish (es)
Bulgarian (bg)	Japanese (ja)	Swedish (sv)
Canadian English	Korean (ko)	Tamil (ta)
Catalan (ca)	Lithuanian (lt)	Traditional Chinese
Czech (cs)	Norwegian Bookmal (nb)	Turkish (tr)
Danish (da)	Norwegian bokmaal (no)	Ukrainian (uk)
Dutch (nl)	Panjabi (pa)	Welsh (cy)
Finnish (fi)		

27th Internationalization and Unicode Conference 7 Berlin, Germany, April 2005

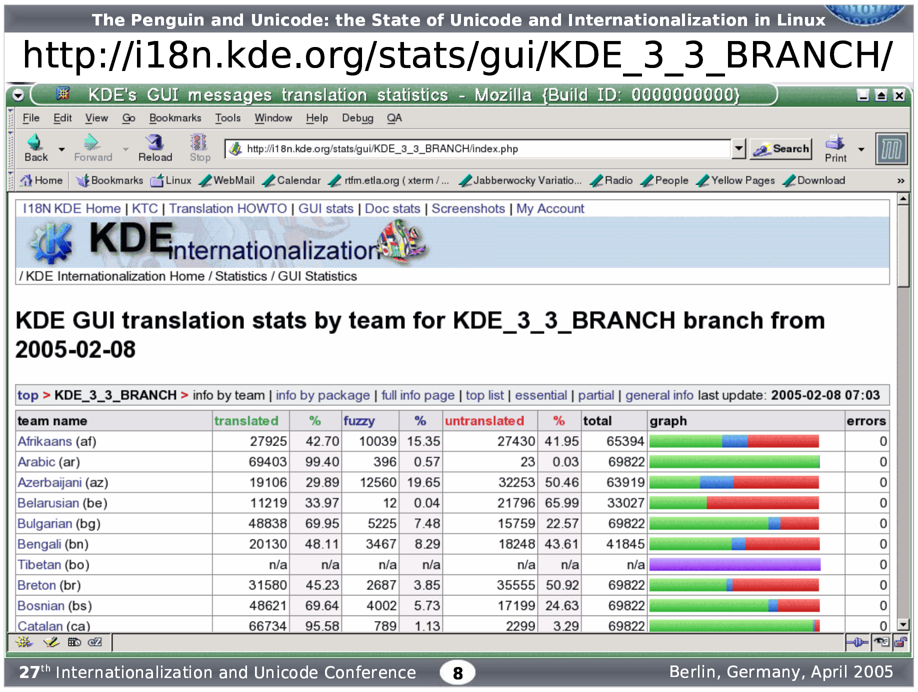
This list displays the 40 most complete language localizations in the Gnome 2.8 desktop as of January, 2005. All of these localization projects are more than 80% complete. Note the presence of Basque, a language of only 588,000 speakers[1], and Welsh with only 580,000 speakers.

Also note the presence of Indonesian. The internet was introduced into Indonesia in 1994, and internet penetration in Indonesia in 2000 was only about 0.1%[2].

Now let's take a look at the KDE project ...

1. All statistics on the number of speakers of various languages are based on data obtained from the United Nations Office of the High Commissioner for Human Rights (UNHCHR) **Universal Declaration of Human Rights** web pages at http://www.unhcr.ch/udhr/navigate/alpha.htm.

2. Zuraida Boerhandoeddin, **E-Commerce in Indonesia**, http://www.isoc.org/inet2000/cdproceedings/7c/7c_3#s2.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

http://i18n.kde.org/stats/gui/KDE_3_3_BRANCH/

KDE's GUI messages translation statistics - Mozilla {Build ID: 0000000000}
File Edit View Go Bookmarks Tools Window Help Debug QA
Back Forward Reload Stop
http://i18n.kde.org/stats/gui/KDE_3_3_BRANCH/index.php
Search Print
Home Bookmarks Linux WebMail Calendar rtfm.etla.org (xterm / ... Jabberwocky Variatio... Radio People Yellow Pages Download
I18N KDE Home | KTC | Translation HOWTO | GUI stats | Doc stats | Screenshots | My Account
KDE internationalization
/ KDE Internationalization Home / Statistics / GUI Statistics

KDE GUI translation stats by team for KDE_3_3_BRANCH branch from 2005-02-08

top > **KDE_3_3_BRANCH** > info by team | info by package | full info page | top list | essential | partial | general info last update: **2005-02-08 07:03**

team name	translated	%	fuzzy	%	untranslated	%	total	graph	errors
Afrikaans (af)	27925	42.70	10039	15.35	27430	41.95	65394	🟩🟦🟥	0
Arabic (ar)	69403	99.40	396	0.57	23	0.03	69822	🟩	0
Azerbaijani (az)	19106	29.89	12560	19.65	32253	50.46	63919	🟩🟦🟥	0
Belarusian (be)	11219	33.97	12	0.04	21796	65.99	33027	🟩🟥	0
Bulgarian (bg)	48838	69.95	5225	7.48	15759	22.57	69822	🟩🟦🟥	0
Bengali (bn)	20130	48.11	3467	8.29	18248	43.61	41845	🟩🟦🟥	0
Tibetan (bo)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	🟪	0
Breton (br)	31580	45.23	2687	3.85	35555	50.92	69822	🟩🟦🟥	0
Bosnian (bs)	48621	69.64	4002	5.73	17199	24.63	69822	🟩🟦🟥	0
Catalan (ca)	66734	95.58	789	1.13	2299	3.29	69822	🟩🟦🟥	0

27th Internationalization and Unicode Conference 8 Berlin, Germany, April 2005

KDE Desktop Localization

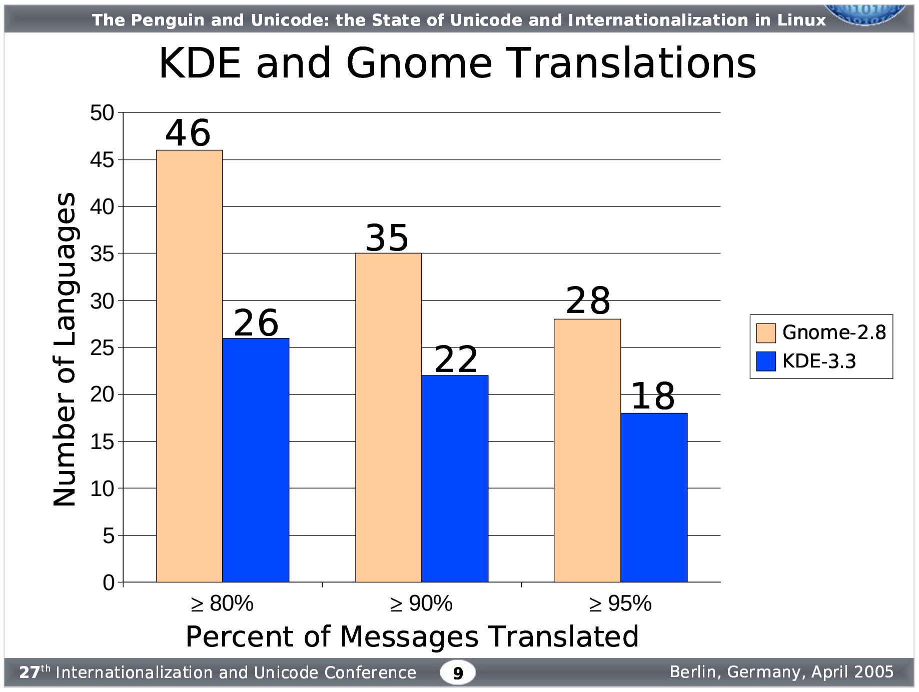
The KDE project has an equivalently lucid web site at http://i18n.kde.org devoted to showing progress on 78 language localization projects (as of Jan. '05). As was true with Gnome, one can easily pick out some interesting facts from these statistics for the KDE 3.3 branch:

- Catalan (Català), spoken by about 4.3 million people in Catalonia, Spain, and Andorra, is more than 95% translated.
- Tajik (Tajiki), a language spoken by nearly 4.4 million people in Tajikistan, Kazakhstan, and other places in Central Asia, is 94% complete.
- Slovenian, spoken by just 2.2 million people, is over 88% complete.
- Icelandic, spoken by just 282,845 people in 2001, is already 77.5% complete.

And in the "honorable mention" category, we have:

- Welsh (Cymraeg, 580,000 speakers) is 51% complete.
- Xhosa, a language of 6.8 million speakers in South and Southeast Africa, is over 48% complete.

An interesting phenomenon in both the KDE and Gnome projects is that languages spoken by relatively small numbers of people in the world, and languages spoken in regions of the world with almost no IT presence until recently (such as Indonesia, Central Asia, and Africa), are beginning to have a fairly impressive presence (judging by message translation statistics) in these Open Source projects.

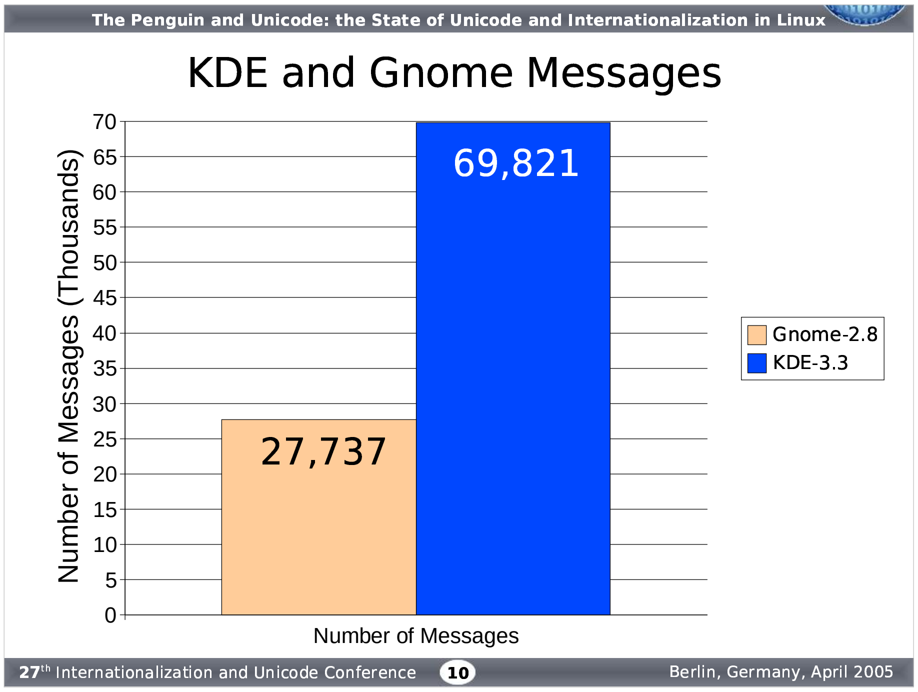


Although both Gnome and KDE have an impressive number of localization teams actively working on translations, from a business perspective it is much more relevant to ask how many of these teams have actually reached a state of 80%, 90% or greater completion?

As the chart above shows, both projects provide a large number of essentially complete and nearly-complete “ready for market” localizations. Note that the statistics shown above are inclusive: that is, the 46 languages complete at the 80% or above level for Gnome (leftmost orange bar) includes the 28 completed at the 95% and above level (rightmost orange bar).

As a point of comparison, in a TechNet article[1] from April, 2003, Edward Ye and Steve Jang of Microsoft define a “fully localized” version of Windows XP as providing “a desktop user with approximately 80% localized user experience.” At the time of that writing, Microsoft Windows XP was available in 24 fully localized versions and in 33 “Language Interface Packs” which install on top of the English version of XP. In contrast, Apple provides complete localizations for 15 languages in OSX.

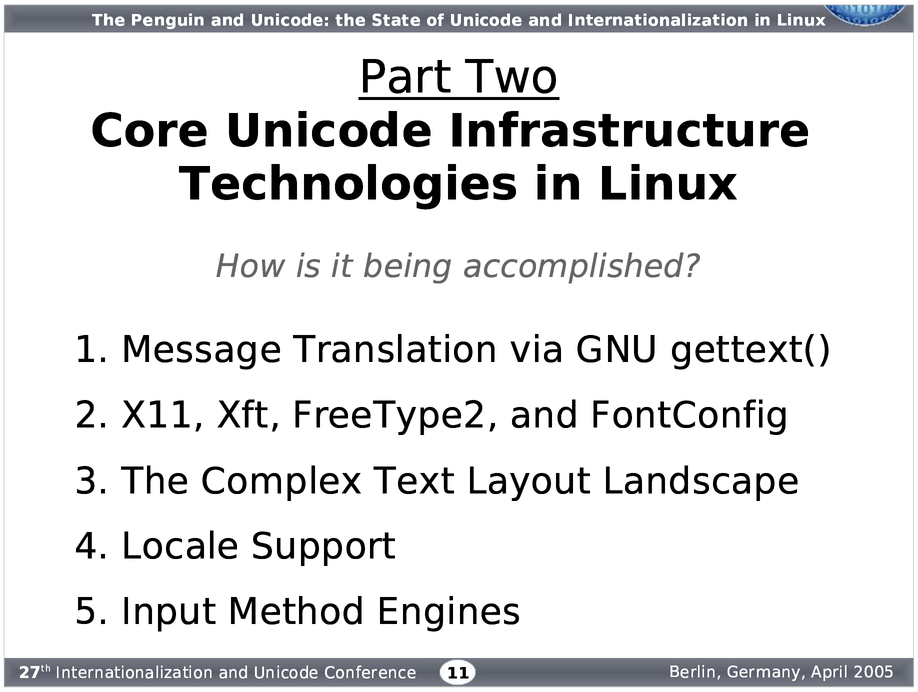
1. Edward Ye and Steve Jang, **Comparing Windows XP Professional Multilingual Options**, http://www.microsoft.com/technet/prodtechnol/winxppro/evaluate/muiovw.mspx#ECAA, 2003.



Since KDE started before Gnome, I wondered how Gnome had managed to quickly achieve support for a greater number of languages at the 80% and above level?

It turns out the answer is fairly straightforward: KDE has nearly 70,000 (69,821) messages in the 3.3 branch. This is a bit more than 2½ times the number of messages found in Gnome version 2.8 (27,737 messages). If you compare how many languages there are in either project localized up to a certain number of messages – say Gnome 2.8's level of almost 28,000 messages-- then you find that the two projects are on an almost equal footing, although Gnome still has a greater number of language localization projects overall.

As we have seen in this brief globalization survey, Linux is, to a large extent, already all over the map. Let's examine, from a technical perspective, how it got there ...



Part Two
Core Unicode Infrastructure Technologies in Linux

How is it being accomplished?

1. Message Translation via GNU gettext()
2. X11, Xft, FreeType2, and FontConfig
3. The Complex Text Layout Landscape
4. Locale Support
5. Input Method Engines

Here in **Part Two** we will look at the Unicode-based infrastructure in Linux that has facilitated the rapid globalization of this Open Source platform.

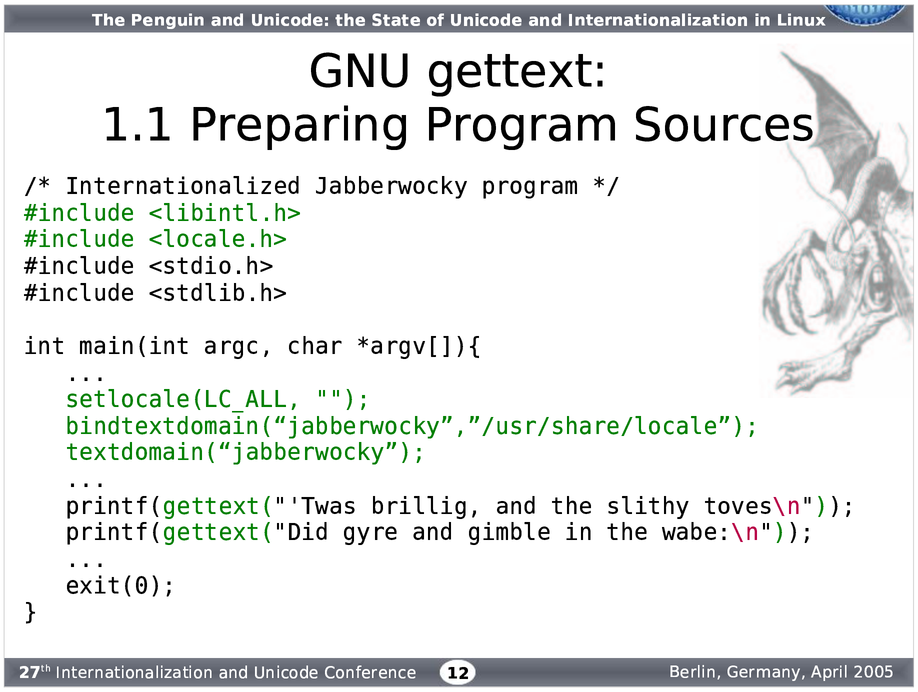
We'll first take a look at message translation via GNU[1] gettext(). We will explore how easy it for developers to use GNU gettext() as a first step in internationalizing their programs.

Next we will look at how X11[2], Xft, FreeType, and FontConfig are wired together to provide font services for modern applications on Linux.

An increasingly important area in today's global market is complex text layout (CTL) needed for scripts like Arabic and Devanagari. We will take a look at the various text layout technologies that are currently available in Linux.

Message translation and internationalized text display are only part of the entire localization equation. End users also expect computers to properly display sorted lists, monetary currencies, numbers, times and calendar dates using local conventions. We will wrap up our review of Linux Unicode technologies by taking a brief look at locale support in the GNU glibc C library[3] and on the desktop. We will also look at one input method engine that has support not only for Chinese, Japanese, and Korean (CJK), but also for other scripts like Amharic.

1. "GNU's not Unix", http://www.gnu.org.
2. The X Windowing system, http://www.x.org, http://www.xfree86.org.
3. GNU glibc: http://www.gnu.org/software/libc/libc.html.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

GNU gettext: 1.1 Preparing Program Sources

```
/* Internationalized Jabberwocky program */
#include <libintl.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
   ...
   setlocale(LC_ALL, "");
   bindtextdomain(“jabberwocky”,”/usr/share/locale”);
   textdomain(“jabberwocky”);
   ...
   printf(gettext("'Twas brillig, and the slithy toves\n"));
   printf(gettext("Did gyre and gimble in the wabe:\n"));
   ...
   exit(0);
}
```

27th Internationalization and Unicode Conference 12 Berlin, Germany, April 2005

The GNU gettext[1] implementation was first implemented by Sun[2] in the Solaris operating system. GNU gettext allows you to internationalize a program in a “minimally invasive” fashion which requires very few changes to existing source code. Virtually all internationalized programs on Linux use GNU gettext.

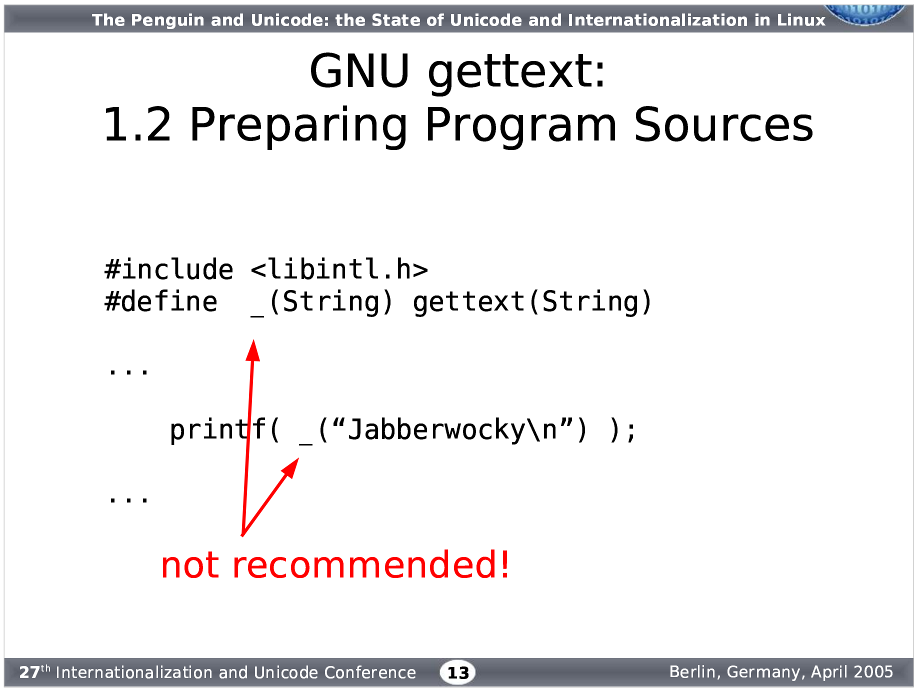
It would be possible to give an entire talk on using the GNU gettext utilities. In this talk, I will limit the discussion to an overview of GNU gettext illustrating how easy it is to use, and leave many of the details as your homework assignment ;-).

Originally I had chosen “Hello, World!” for this example, but Sandra O'Donnell of the conference review board wanted something more “exciting”, so I decided to use Lewis Carroll's famous poem, *The Jabberwocky*[3], which is of course pathologically difficult to translate[4] ...

The first step is to prepare program sources. This is easy to do. First, include the “libintl” and “locale” headers. In main(), add the calls to setlocale(), bindtextdomain(), and textdomain(). Bindtextdomain() tells the library to look for localized message catalogs for the “jabbberwocky” program in the “/usr/share/locale” directory. textdomain() sets the text domain. After that, all that is required is to wrap all translatable strings with calls to gettext().

For those of you familiar with using Trolltech's QT toolkit, you may notice that the use of gettext() is just like the use of Qobject::tr() in QT. Note however that KDE, even though it is built on top of QT, uses GNU gettext() rather than the QT linguist functions.

1. **GNU gettext Utilities**, http://www.gnu.org/software/gettext/manual/gettext.html. 2. **Sun Microsystems**, http://www.sun.com. 3. Found in **Through The Looking Glass**, by Lewis Carroll, first published in December, 1871. 4. Translating Jabberwocky: http://www76.pair.com/keithlim/jabberwocky/poem/hofstadter.html .

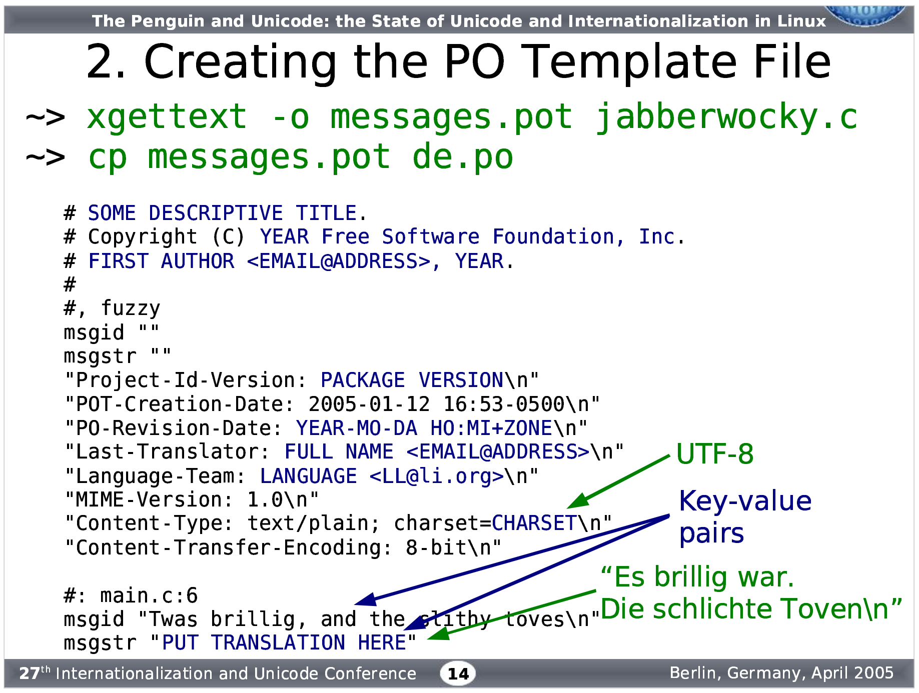


GNU gettext: 1.2 Preparing Program Sources

```
#include <libintl.h>
#define  _(String) gettext(String)

...

    printf( _("Jabberwocky\n") );

...
```

not recommended!

The GNU gettext manual mentions that some GNU packages use a simple "_" underscore internally, so you might see ' _("Jabberwocky")' instead of ' gettext ("Jabberwocky")'. I would not recommend this practice to anyone writing new programs, since it may be unclear to readers of the code what the "_" underscore is supposed to mean, and code clarity is an imperative in any well-written program.



2. Creating the PO Template File

```
~> xgettext -o messages.pot jabberwocky.c
~> cp messages.pot de.po
```

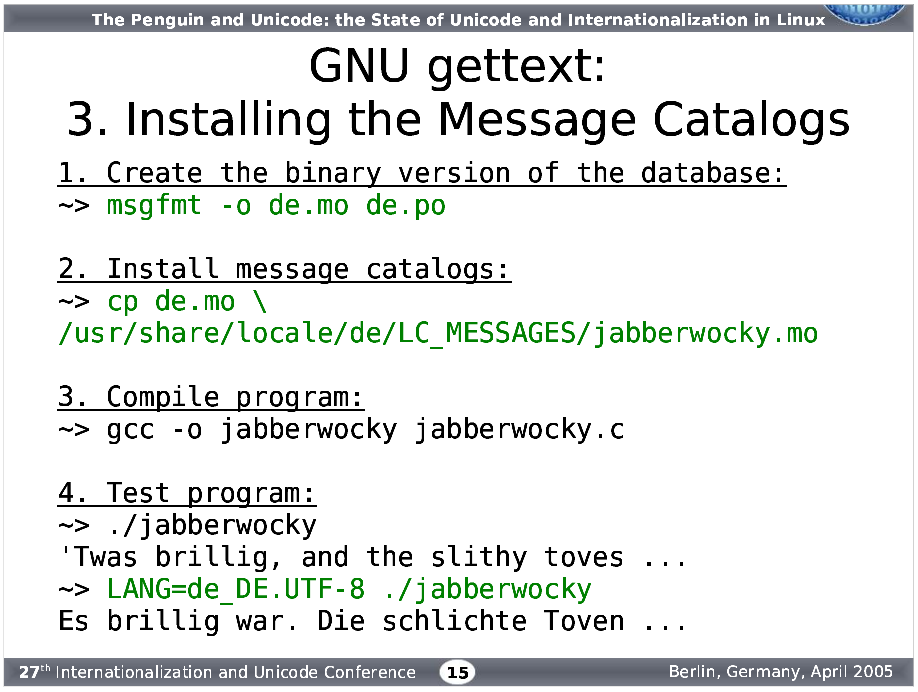
```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR Free Software Foundation, Inc.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 2005-01-12 16:53-0500\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8-bit\n"

#: main.c:6
msgid "Twas brillig, and the slithy toves\n"
msgstr "PUT TRANSLATION HERE"
```

After you have modified your program sources, the xgettext utility allows you to extract the translatable strings into a “portable object template” (POT) file. This file becomes the template for your various translation files. For example, after filling in the relevant information regarding your package (in blue), you would copy the POT file to “de.po” to use for the German translation. The “PO” extension indicates that the database is in a portable, human-readable format.

Although the GNU glibc internationalization library allows you to use any number of valid ISO and other national encoding standards, to ensure that your translation files are truly portable and readable by everyone, I highly recommend that you always use UTF-8 as the encoding format. Major Linux distributions like Novell/SuSE and Redhat are already using UTF-8 locales by default in almost all cases, and therefore you should too.

The translation database itself simply consists of a series of key-value pairs, where the msgid in English is the key, and the msgstr in the target language (German in this case) is the value for that key.

In the third part of this talk we will take a look at **KBabel** which gives you a graphical environment for editing and managing PO translation files.



GNU gettext: 3. Installing the Message Catalogs

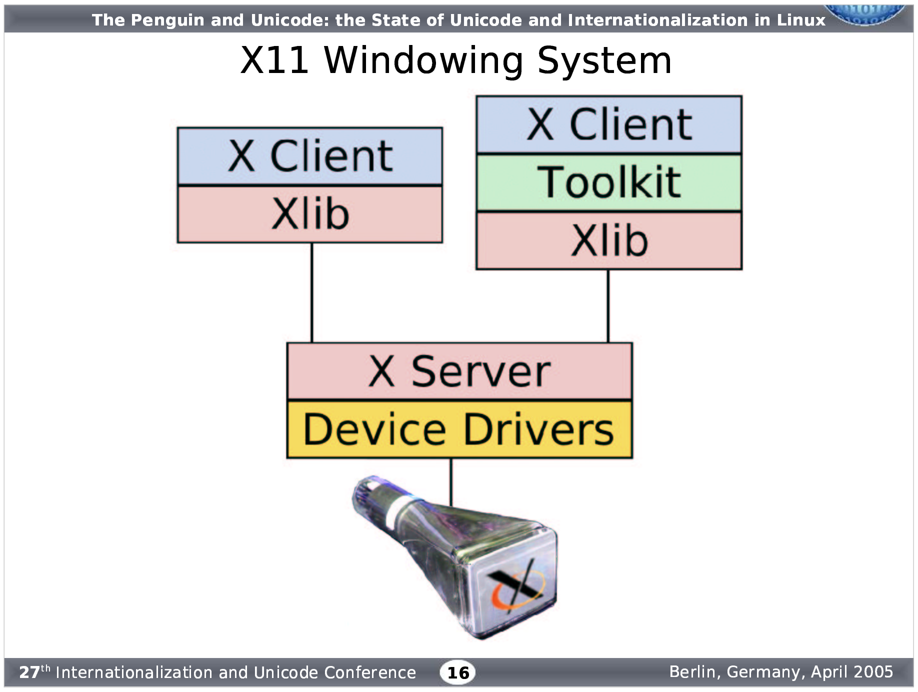
```
1. Create the binary version of the database:
~> msgfmt -o de.mo de.po

2. Install message catalogs:
~> cp de.mo \
/usr/share/locale/de/LC_MESSAGES/jabberwocky.mo

3. Compile program:
~> gcc -o jabberwocky jabberwocky.c

4. Test program:
~> ./jabberwocky
'Twas brillig, and the slithy toves ...
~> LANG=de_DE.UTF-8 ./jabberwocky
Es brillig war. Die schlichte Toven ...
```

Once you have completed translating your message catalogs, the msgfmt utility is used to create binary “machine object” (MO) versions of your catalogs. Once that is done, all that is left is to install the message catalogs and test the program.

In real projects on Linux, the paths where message catalogs will be installed are configured when you run autoconf's[1] ./configure script. Usually the ./configure script is responsible for creating a config.h header file that contains the actual path that will be passed to bindtextdomain(). Additionally, in a real project you would run msgfmt and the installation routines from a project's Makefile.

Now that we understand how to get localized text messages into a Linux program, let's examine the infrastructure underlying the display of those messages on the desktop.

1. **Autoconf** is a tool used to create shell scripts used for automatically configuring source code software packages on Linux/Unix systems: http://www.gnu.org/software/autoconf.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

X11 Windowing System

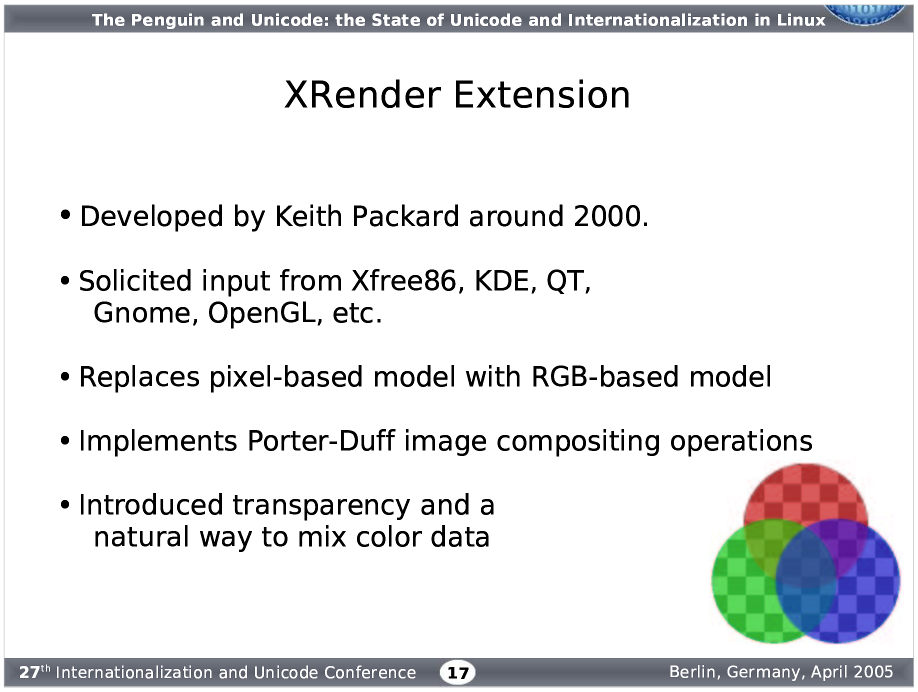
27th Internationalization and Unicode Conference 16 Berlin, Germany, April 2005

The X Protocol was developed in the mid 1980's to provide a network-transparent graphical user interface (GUI) for the UNIX operating system. The X server represents the display server, while X clients are software applications started by the user. Although X clients may talk to Xlib directly (as illustrated on the left), it is now much more common to use a GUI toolkit such as GTK+[1] or QT[2] (right).

Historically, the X client application would typically run on a remote machine, such as a computational server which had greater computing power than the local display machine. Nowadays, with the ubiquity of PCs, applications are often run on the same machine as the X server. Regardless of whether an application is run remotely or locally, X uses an asynchronous network protocol for communication between the X client and the X server. All the details of the hardware and operating system are hidden from the application. This greatly simplifies the development of X client applications and makes X-based apps highly portable across numerous operating systems. This portability is one reason for X Windows continuing appeal –"resurgence" might be more accurate -- despite the age of the technology.

Although network transparency and application portability have continued to be highly desired, other aspects of X Windows, such as the original graphics drawing and font-handling systems, could not meet the demands of modern desktops like KDE and Gnome. As a result, a number of extensions to X have been created to resolve serious shortcomings in the original technology. In the next few slides, we will look at the XRender and XFT extensions which are partially responsible for the resurgence of X and the wider acceptance of Linux as a desktop OS.

1. The Gimp Toolkit, http://www.gtk.org. The Gimp toolkit is used in both Gnome and Firefox.
2. Trolltech's QT toolkit, http://www.trolltech.com. QT is used as the foundation of the KDE desktop.



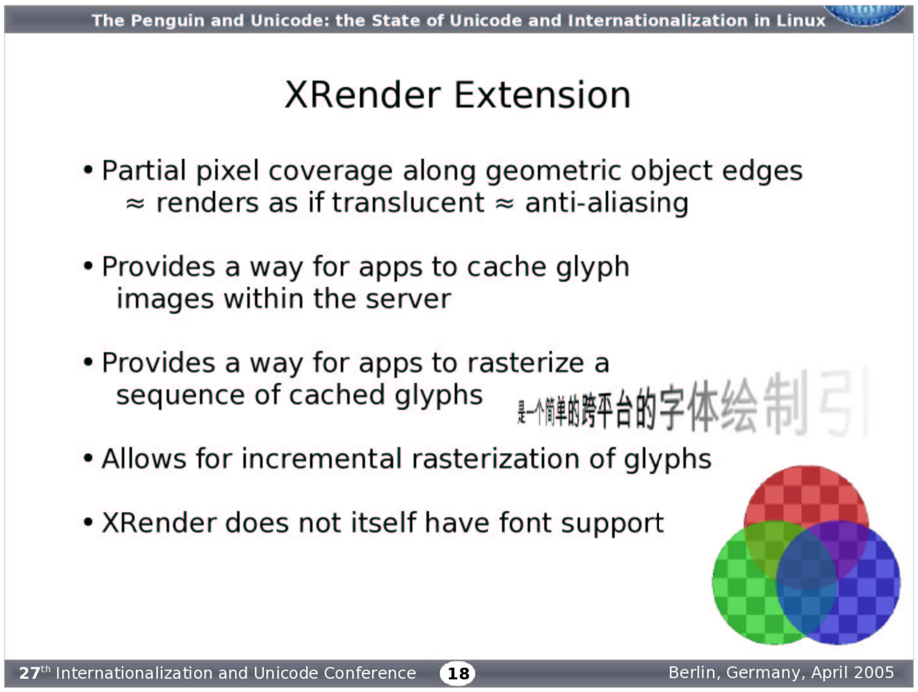
XRender Extension

- Developed by Keith Packard around 2000.
- Solicited input from Xfree86, KDE, QT, Gnome, OpenGL, etc.
- Replaces pixel-based model with RGB-based model
- Implements Porter-Duff image compositing operations
- Introduced transparency and a natural way to mix color data

XRender[1] was developed by Keith Packard around 2000. Keith solicited input from numerous players in the X windowing system community, including developers in the XFree86, QT, KDE, GTK, Gnome, and OpenGL projects.

The XRender extension replaced the pixel-based model of rendering graphics with an RGB-based model. The extension also implemented the compositing operations of Thomas Porter and Tom Duff[2]. This introduced transparency and a natural way to mix colors.

1. Keith Packard, **Design and Implementation of the X Rendering Extension**, http://keithp.com/~keithp/talks/usenix2001/.
2. Thomas Porter and Tom Duff. **Compositing Digital Images**. Computer Graphics 18(3):253-259, July 1984.



XRender Extension

- Partial pixel coverage along geometric object edges ≈ renders as if translucent ≈ anti-aliasing
- Provides a way for apps to cache glyph images within the server
- Provides a way for apps to rasterize a sequence of cached glyphs
- Allows for incremental rasterization of glyphs
- XRender does not itself have font support

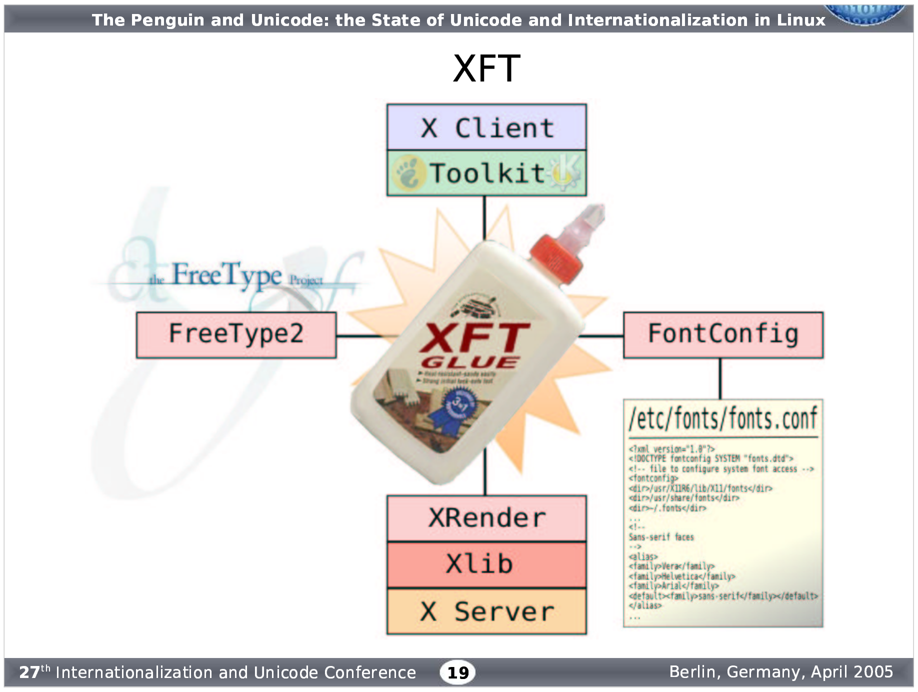
Since XRender introduced transparency, it was now possible to treat partial pixels along the edges of geometric objects as if they were translucent and thus approximate anti-aliasing.

Also, XRender provided a way for applications to cache glyph images within the X server (in a GlyphSet) and then to rasterize a sequence of cached glyphs. Each Glyph in a GlyphSet is essentially a Picture with additional geometric positioning and glyph advance information. XRender also allowed for the incremental rasterization of glyphs, thus eliminating the inefficiencies of rasterizing every glyph in a CJK font, for example.

When XRender was first developed, there was concern about the increase in network traffic caused by the need for clients to send rasterized glyph images to the server for caching. However, tests proved that the glyph image traffic was more than offset by the fact that the X server no longer needed to send font names and glyph metrics to the client.

Note that Xrender does not itself have font support: applications are responsible for locating and rasterizing glyphs and obtaining the geometric information associated with glyphs.

This led Keith Packard and others to develop Xft ...



The Penguin and Unicode: the State of Unicode and Internationalization in Linux

XFT

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<!-- file to configure system font access -->
<fontconfig>
<dir>/usr/X11R6/lib/X11/fonts</dir>
<dir>/usr/share/fonts</dir>
<dir>~/.fonts</dir>
...
<!--
Sans-serif faces
-->
<alias>
<family>Vera</family>
<family>Helvetica</family>
<family>Arial</family>
<default><family>sans-serif</family></default>
</alias>
...
```

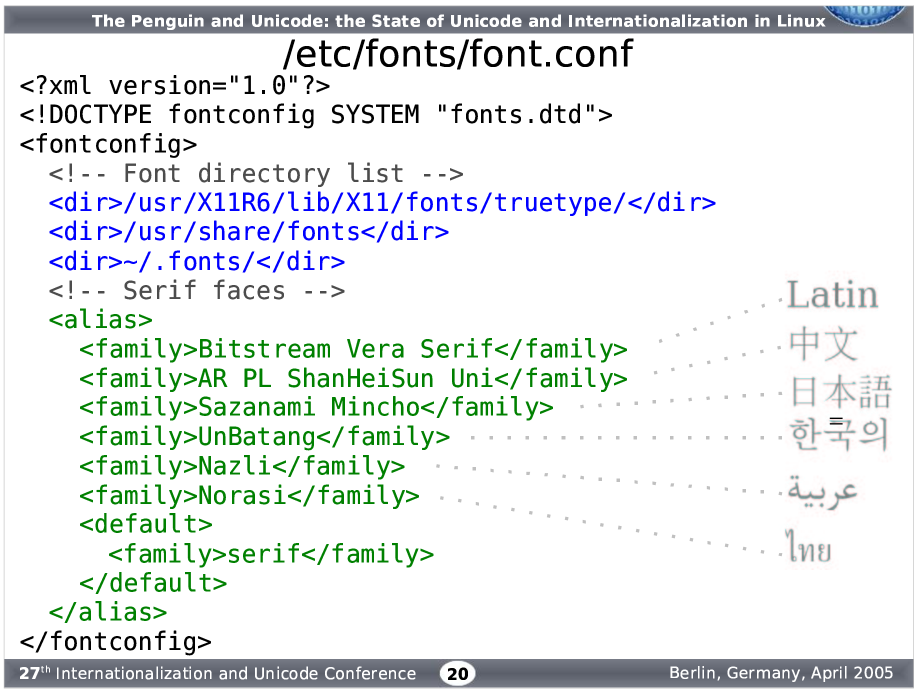
27th Internationalization and Unicode Conference 19 Berlin, Germany, April 2005

David Turner and others had developed an Open Source font rasterizer called "FreeType"[1]. FreeType was already a fairly mature project when Keith Packard started working on Xft. FreeType2, the successor to the original FreeType library, was designed to be compact, efficient, and portable across all kinds of systems, including embedded systems. FreeType2 currently supports over 11 different font formats, including both the TrueType and CFF variants of OpenType and font formats used on X11 such as PCF and BDF formats.

Xft[2] provides the glue that allows X client applications to interface to the FreeType2 font rasterizer and to the XRender extension. In order to greatly simplify the design of the library, all text output routines accept only Unicode-encoded data. Since neither FreeType nor the XRender extension provided any kind of font configuration or customization mechanisms, Xft originally handled this. Later, during a redesign of Xft, the font configuration and customization portions of the library were broken out into a separate library called FontConfig.

FontConfig[3] permits applications to locate fonts installed on the system and uses sophisticated font matching and selection algorithms to identify fonts according to certain characteristics, such as family, weight, or Unicode range. FontConfig also provides mechanisms for customizing fonts, for example by mapping generic names like "sans-serif", "serif", and "monospace" to fonts available on the system. Configuration information is stored in XML-based files, such as the system-wide /etc/fonts/fonts.conf file.

1. Excellent documentation on FreeType is available at http://www.freetype.org.
2. Keith Packard, **The Xft Font Library: Architecture and Users Guide**, http://keithp.com/~keithp/talks/xtc2001, 2001.
3. Keith Packard, **Font Configuration and Customization for Open Source Systems**, http://keithp.com/~keithp/talks/guadec2002, 2002.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

/etc/fonts/font.conf

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<fontconfig>
  <!-- Font directory list -->
  <dir>/usr/X11R6/lib/X11/fonts/truetype/</dir>
  <dir>/usr/share/fonts</dir>
  <dir>~/.fonts/</dir>
  <!-- Serif faces -->
  <alias>
    <family>Bitstream Vera Serif</family>
    <family>AR PL ShanHeiSun Uni</family>
    <family>Sazanami Mincho</family>
    <family>UnBatang</family>
    <family>Nazli</family>
    <family>Norasi</family>
    <default>
      <family>serif</family>
    </default>
  </alias>
</fontconfig>
```

Latin
中文
日本語
한국의
عربية
ไทย

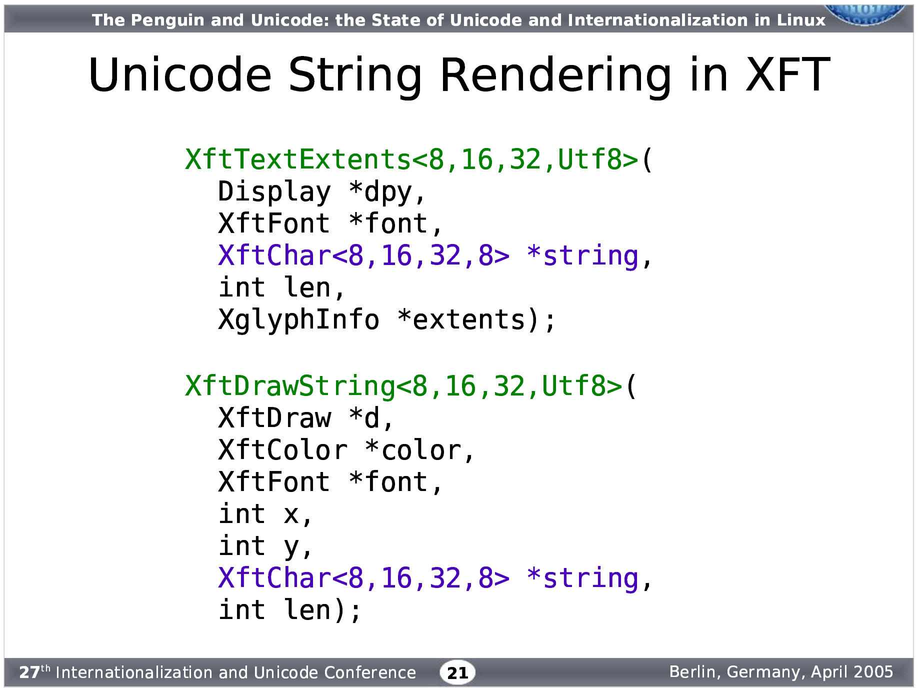
27th Internationalization and Unicode Conference 20 Berlin, Germany, April 2005

Here is a truncated version of the global /etc/fonts/fonts.conf XML file.

At the top specifications for font directory search paths are enumerated.

Aliases for mapping the generic category of “serif” fonts to fonts actually available on the system are shown next. The font names shown in this example are open source Unicode fonts (Vera is a Latin font, ShanHeiSun is a Chinese font, Sazanami Mincho is a Japanese font, UnBatang is a Korean font, Nazli is an Arabic/Farsi font, and Norasi is a Thai font). XFT/Fontconfig-aware software, such as the Mozilla/Firefox web browser, can automatically use the fonts listed here when, for example, a CSS style sheet attached to an HTML document specifies a generic “serif” font.

A complete configuration file normally specifies many more options than can be shown here. To begin with, a real configuration file would also have specifications for sans-serif and monospace fonts. The file might also specify options for sub-pixel rendering that are appropriate for LCD monitors. Also there is usually a block that specifies a transformation matrix to use for artificially obliquing fonts which lack real italic or oblique versions. Finally, the global /etc/fonts/fonts.conf file normally also contains a line for including a user-specific ~/.fonts.conf file located in the user's home directory so that individual users may customize font behaviour according to their own liking.



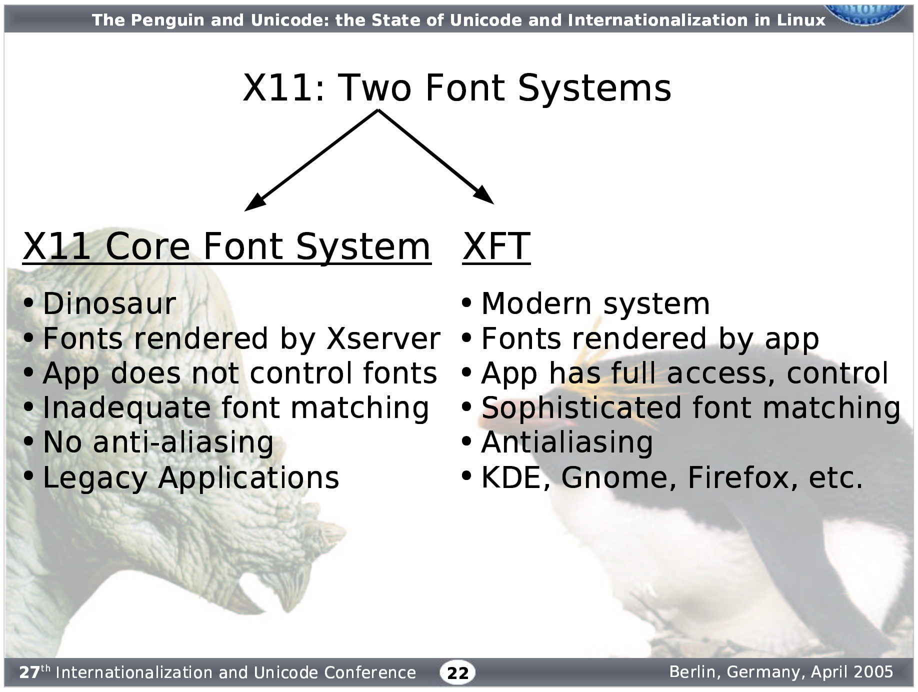
Unicode String Rendering in XFT

```
XftTextExtents<8,16,32,Utf8>(
  Display *dpy,
  XftFont *font,
  XftChar<8,16,32,8> *string,
  int len,
  XglyphInfo *extents);

XftDrawString<8,16,32,Utf8>(
  XftDraw *d,
  XftColor *color,
  XftFont *font,
  int x,
  int y,
  XftChar<8,16,32,8> *string,
  int len);
```

As mentioned, XFT accepts only Unicode-encoded strings. The first question in the minds of developers is “which Unicode encoding is required?”. As you can see here, the developers of XFT thoughtfully allow you to use whichever Unicode transformation format you want.

For many projects on Linux, the *de facto* standard is to use UTF-8. Linux is based on Unix which has historically used byte-based data streams. The design and subsequent use of UTF-8 on Unix allowed many Unix utilities to handle Unicode data streams with minimal changes to the existing byte-oriented code. UTF-8 continues to be the preferred Unicode transformation format on Linux today for numerous reasons[1] which tend to simplify things for the Linux programmer. Thus, many projects on Linux make use of XftDrawStringUtf8() which accepts an ordinary ANSI C string containing UTF-8 text. However, if you need to, you can use XftDrawString16() to process UTF-16 strings, or XftDrawString32() to processs UCS-4 strings.

1. Advantages of UTF-8 include, but are not limited to: ① ASCII characters are transported transparently in UTF-8 streams, ② Regardless of whether you begin parsing at the start, middle, or end of a UTF-8 string, you can always quickly locate the lead bytes which serve as synchonization points in a string, ③ Comparing UTF-8 strings using ANSI C strcmp() preserves Unicode canonical sorting just as if you had done the string comparison using wcscmp() on UCS-4 strings, ④ no byte order mark (BOM) is required. One can enumerate additional arguments in favor of UTF-8 for Linux and Unix platforms: see Google's cached copy of Roman Czyborra's page on Unicode transformation formats, http://64.233.167.104/search?q=cache:Kw7QqNNqjaUJ:czyborra.com/utf/.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

X11: Two Font Systems

X11 Core Font System
- Dinosaur
- Fonts rendered by Xserver
- App does not control fonts
- Inadequate font matching
- No anti-aliasing
- Legacy Applications

XFT
- Modern system
- Fonts rendered by app
- App has full access, control
- Sophisticated font matching
- Antialiasing
- KDE, Gnome, Firefox, etc.

27th Internationalization and Unicode Conference 22 Berlin, Germany, April 2005

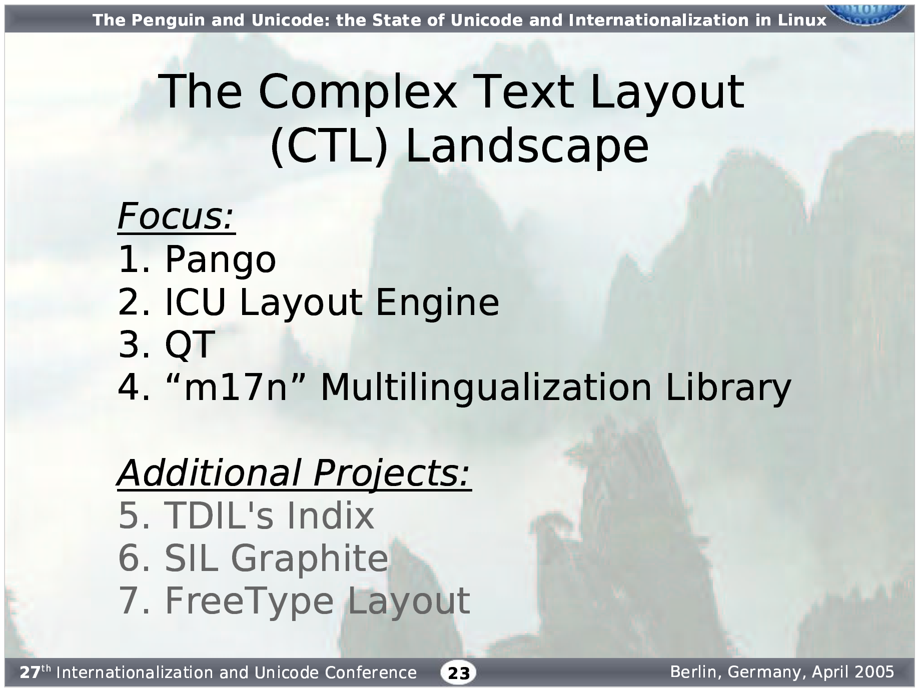
As a result of the work of Keith Packard, David Turner, and many others, X11 now has two font systems: (1) the original Core Font System, and (2) the modern Xft system. Taken together, XRender, Xft, FreeType, and FontConfig have solved a host of problems that previously plagued the X windowing system.

Applications now have full access to font files, and thus the full control needed for advanced typography such as kerning and ligature substitution. Fontconfig provides sophisticated font pattern matching. XRender supports antialiasing on modern X servers such as X.org and XFree86, but is smart enough to fall back to the old X Core rendering when talking to X servers that do not support the Render extension.

Both Trolltech's QT toolkit and the Gimp GTK+ toolkit use Xft. As a result, the two most popular Linux desktop environments, KDE (built on top of QT) and Gnome (built using GTK+), as well as a host of other important modern applications such as the Mozilla and Firefox web browsers, all take advantage of Xft.

Developers wishing to build modern applications for the Linux operating system no longer need to be constrained by the historical X Core font system. By using a modern toolkit like QT or GTK+, or an application framework like Gnome or KDE, developers have complete access to modern font services.

While Xft along with FreeType2 and FontConfig provide ample *font services*, we still need to look at the question of *text services*, especially the problem of complex text layout (CTL) services needed for a host of important world scripts, including such scripts as Arabic, Syriac, Devanagari, Bengali, Thai, Khmer, Tibetan, and so on ...


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

The Complex Text Layout (CTL) Landscape

Focus:
1. Pango
2. ICU Layout Engine
3. QT
4. “m17n” Multilingualization Library

Additional Projects:
5. TDIL's Indix
6. SIL Graphite
7. FreeType Layout

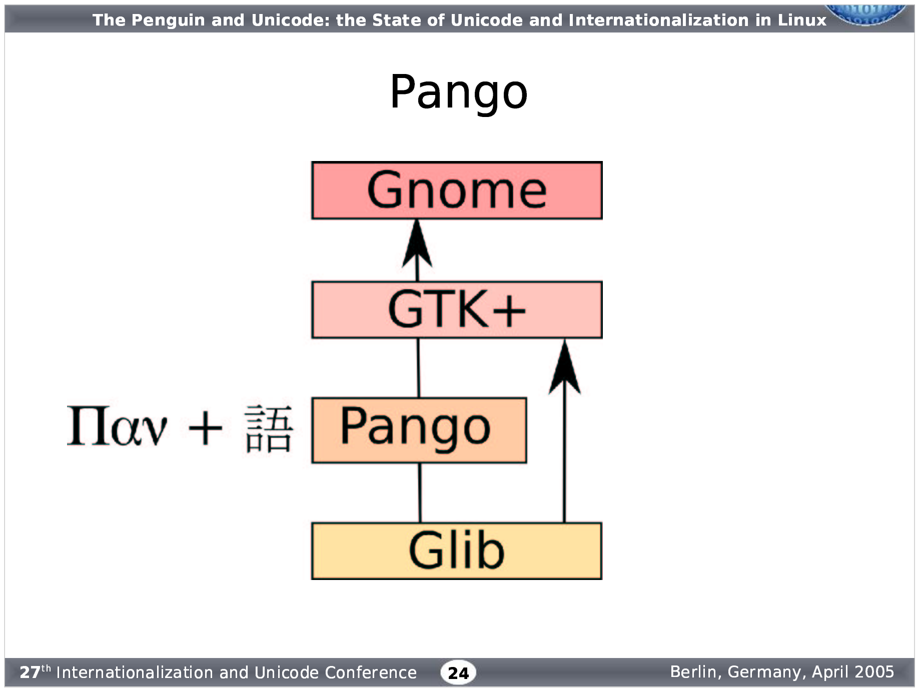
27th Internationalization and Unicode Conference 23 Berlin, Germany, April 2005

On Apple OS X, there is ATSUI[1]. On Windows, there is Uniscribe[2]. End of story! But on Linux, it is quite a different story! On Linux, there are: (1) Owen Taylor's Pango library, (2) Eric Mader's Layout Engine in IBM's International Components for Unicode (ICU), (3) Trolltech's QT toolkit, and (4) Japan's National Institute of Advanced Science and Technology's (AIST) “m17n” multilingual layout library. In addition to these four, there are also (5) India's TDIL Indix project[3], (6) SIL International's Graphite[4] project, and, finally, (7) the still vaporous FreeType Layout[5] project.

The Complex Text Layout landscape in Linux is ... complex! Different software can exhibit differing levels of support for CTL scripts like Arabic or Kannada. Some scripts, like Burmese and Khmer, are currently hardly supported at all in the majority of software on Linux today. The situation is confusing if you are a software developer or provider. It is even more confusing if you are an end-user of Linux trying to fathom why, for example, Kannada is rendered correctly in a Gnome program, but shows rendering artifacts in OpenOffice even when using the very same OpenType font in both software packages.

In this talk, we focus on the top four projects listed above. We could use many different metrics to compare these projects. For example, developers might want to know whether the software is written in C or in C++. It would also be helpful to know whether the documentation is good or poor. However, the most important metric in my mind is simply how many different CTL scripts does the layout engine actually support? In addition to introducing these four projects, I will be presenting which CTL scripts are currently supported, or will soon be supported, in these libraries.

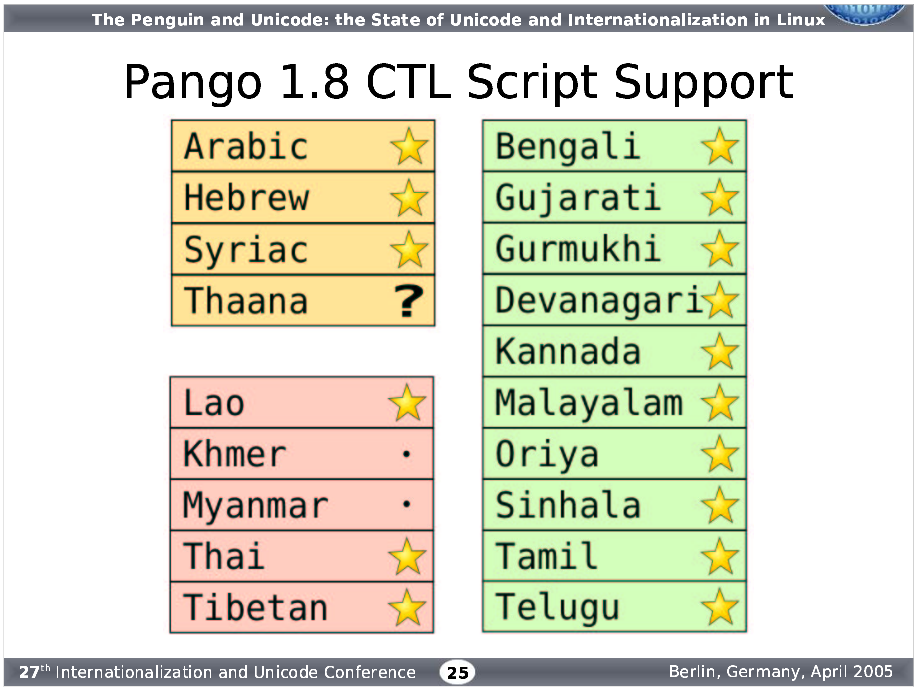
1. Apple Type Services for Unicode Imaging, http://developer.apple.com/intl/atsui.html. 2. Uniscribe, http://www.microsoft.com/typography/developers/uniscribe. 3. Indix, http://tdil.mit.gov.in/download/INDIX.htm. 4. SIL Graphite, http://scripts.sil.org/. 5. FreeType Layout, http://freetype.sourceforge.net/ftlayout/.



Pango

Pango[1] is the internationalized text layout library written in C by Owen Taylor for the Gnome project. The name “Pango” derives from the Greek “παν” meaning “all”, and the Japanese “ 語 ” “ go” meaning language. Although Pango is used in the Gnome project, Pango is only dependent on the low-level glib library (part of the GTK+ project). No other GTK+/Gnome libraries are required. Pango can be built with backends to FreeType, X, and Xft. Thus it is fairly easy to use Pango in other projects that are not related to the Gnome desktop. For example, we are currently using Pango as the layout engine for the LASi[2] Unicode Postscript printing library in Dr. Richard's lab at the Kellogg Eye Center.

1. Pango, http://www.pango.org.
2. LASi, http://eyegene.ophthy.med.umich.edu/lasi/.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

Pango 1.8 CTL Script Support

Arabic	☆	Bengali	☆
Hebrew	☆	Gujarati	☆
Syriac	☆	Gurmukhi	☆
Thaana	?	Devanagari	☆
		Kannada	☆
Lao	☆	Malayalam	☆
Khmer	•	Oriya	☆
Myanmar	•	Sinhala	☆
Thai	☆	Tamil	☆
Tibetan	☆	Telugu	☆

27th Internationalization and Unicode Conference 25 Berlin, Germany, April 2005

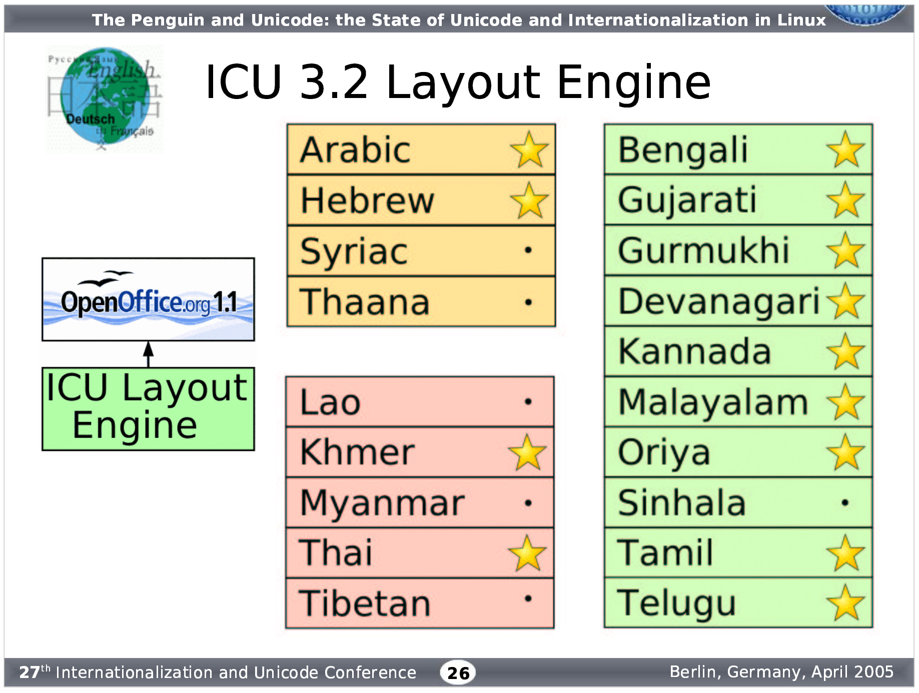
Pango version 1.8 was released in December of 2004. In this chart, stars indicate full OpenType support for scripts in Pango version 1.8.

Eric Mader, the author of ICU's Layout Engine, ported OpenType support for the majority of Indic scripts to Pango about two years ago.

The Arabic support includes full OpenType support for vowel marks and positioning for Nashtaliq slanted baselines needed for languages like Urdu. Hebrew support includes OpenType support to position cantillation marks.

Release 1.8 added support for Syriac, Sinhala, Lao, and Tibetan scripts. The KhmerOS project[1] is working on Khmer support in Pango. As of this writing (Feb. '05) the Khmer module was in the process of being reviewed and revised prior to incorporation into the next release of Pango. The next release of Pango is scheduled for mid-2005, so it is likely that Khmer support will be included in Pango 1.10. Thaana is an RTL script, but is not cursive. Since Pango has OpenType support for Hebrew, it is possible that Thaana may work in Pango but I have not been able to confirm this.

1. KhmerOS is a project to bring Khmer to computers, http://www.khmeros.info.



ICU 3.2 Layout Engine

OpenOffice.org 1.1
ICU Layout Engine

Arabic	☆
Hebrew	☆
Syriac	•
Thaana	•
Lao	•
Khmer	☆
Myanmar	•
Thai	☆
Tibetan	•
Bengali	☆
Gujarati	☆
Gurmukhi	☆
Devanagari	☆
Kannada	☆
Malayalam	☆
Oriya	☆
Sinhala	•
Tamil	☆
Telugu	☆

IBM's International Components for Unicode[1] version 3.2 Layout Engine (released Nov. 2004) includes OpenType layout support for the scripts with stars shown above, with the exception of Thai. Thai is supported, but Thai OpenType tables are not yet supported. As is the case with Pango, the Arabic support includes full OpenType support for Nashtaliq.

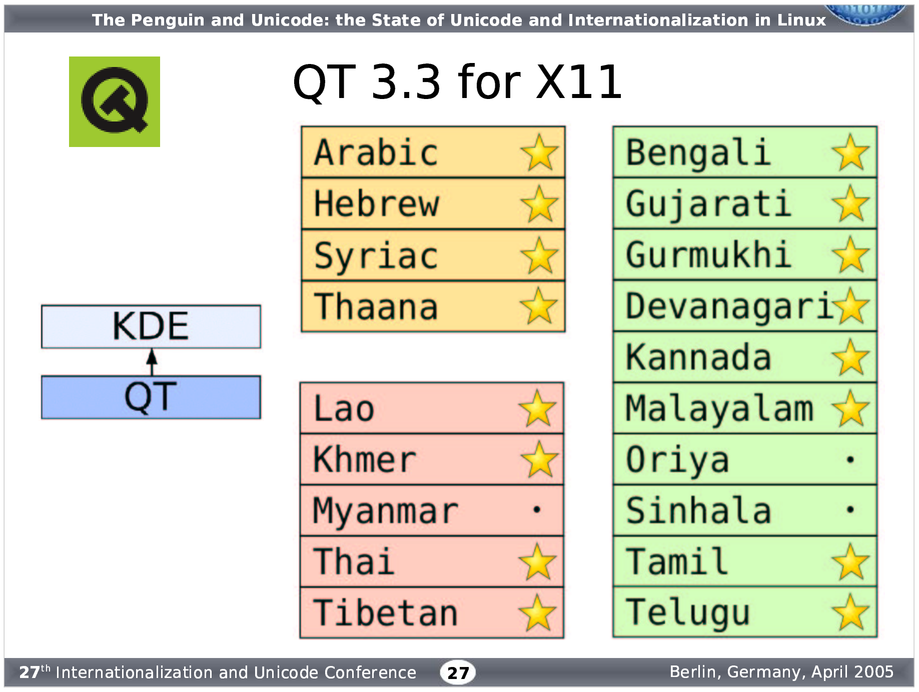
Eric Mader, the author of ICU's Layout Engine, told me that some bugs probably still remain in the Indic scripts. The Khanda Ta[2] problem is a known problem in Bengali Unicode support which requires resolution from the Unicode Consortium.

As mentioned, Eric Mader was responsible for porting Indic OpenType support to Pango a few years ago. Since that time, Owen Taylor of RedHat and Mader have collaborated informally in synchonizing bug fixes between Pango and ICU. As a result, users should expect to see very similar results from both Pango and ICU where these engines have overlapping script support.

ICU just got Khmer support at the end of December, 2004. Support for Lao, Myanmar, Tibetan, and Sinhala is still missing. Mader expects that porting Pango's Sinhala support to ICU will probably not be much trouble.

Note that on Linux, the OpenOffice suite uses ICU's Layout Engine, although what was used at the time of this writing was an older 2.x version, not the latest 3.2 version. On Windows, OpenOffice uses Uniscribe.

1. International Components for Unicode, http://oss.software.ibm.com/icu/.
2. Peter Constable, Encoding of Bengali Khanda Ta in Unicode, Public Review Issue #30, http://www.unicode.org/review/pr-30.pdf.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

QT 3.3 for X11

Arabic	☆
Hebrew	☆
Syriac	☆
Thaana	☆
Lao	☆
Khmer	☆
Myanmar	•
Thai	☆
Tibetan	☆

Bengali	☆
Gujarati	☆
Gurmukhi	☆
Devanagari	☆
Kannada	☆
Malayalam	☆
Oriya	•
Sinhala	•
Tamil	☆
Telugu	☆

27th Internationalization and Unicode Conference 27 Berlin, Germany, April 2005

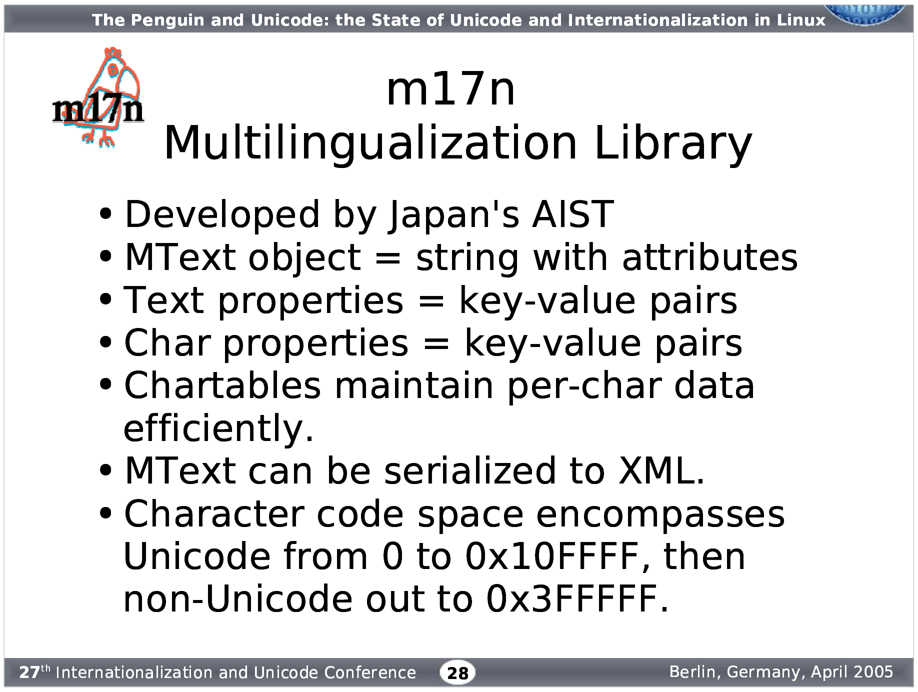
QT[1] is a cross-platform C++ application framework released by Trolltech. Open Source versions for the X11 platform are available for those creating Open Source software.

The KDE desktop uses QT as its base technology. This table shows CTL scripts supported by QT version 3.3 on the X11 platform at the time of this writing (Jan. '05). Script support on Windows or Apple OSX may be slightly different.

Support for the complex scripts shown above (and of course other non-complex scripts present in Unicode) is completely transparent to the programmer. The programmer can, for the most part, just use QT's QString, QLineEdit, QTextEdit, QLabel, and derived classes. As QT provides a complete object-oriented application framework that is also cross-platform, it certainly represents a very attractive option for software developers and businesses looking to expand into the Linux market.

In contrast, IBM's ICU is platform-independent but cannot directly display text without having a concrete rendering class written on top of the platform-agnostic base class.

1. Trolltech's QT, http://www.trolltech.com.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

m17n Multilingualization Library

- Developed by Japan's AIST
- MText object = string with attributes
- Text properties = key-value pairs
- Char properties = key-value pairs
- Chartables maintain per-char data efficiently.
- MText can be serialized to XML.
- Character code space encompasses Unicode from 0 to 0x10FFFF, then non-Unicode out to 0x3FFFFF.

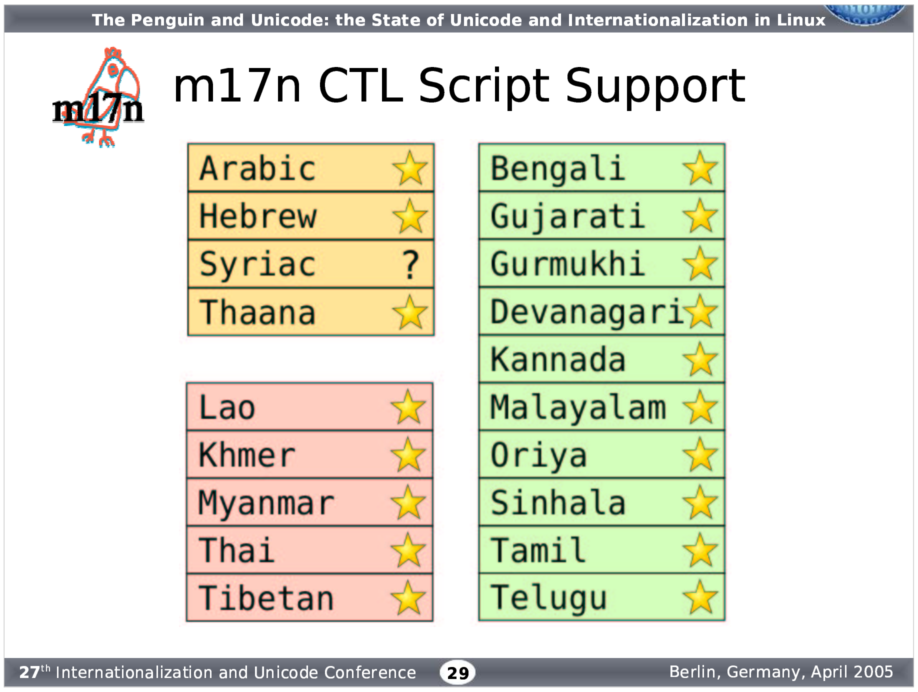
27th Internationalization and Unicode Conference 28 Berlin, Germany, April 2005

The “m17n”[1] Multilingualization library is a project developed by Japan's National Institute of Advanced Industrial Science and Technology (AIST). The library is written in C. The library has many interesting features.

Multilingual text is represented using an “MText” object. An MText object is basically a string with attributes called text properties. The MText object is designed to be used in place of ordinary C strings in code. An MText can have zero or more text properties. Text properties consist of key-value pairs. In addition to text properties, individual characters can also have properties. Character properties are also key-value pairs. A data structure called a Chartable is used to store per-character properties in an efficient manner. Functions are provided for serializing MText data to and from XML.

The character code space from 0 to 0x10FFFF represents Unicode, while the code space beyond 0x10FFFF all the way out to 0x3FFFFF might be used for processing scripts not yet included in Unicode. Naoto Takahashi, a senior research scientist from AIST, is describing this library in detail in his talk here at the conference entitled “**A Library for Multilingual Text Processing**” (**Session B13**).

1. The m17n Library, http://www.m17n.org.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

m17n CTL Script Support

Arabic	☆
Hebrew	☆
Syriac	?
Thaana	☆
Lao	☆
Khmer	☆
Myanmar	☆
Thai	☆
Tibetan	☆

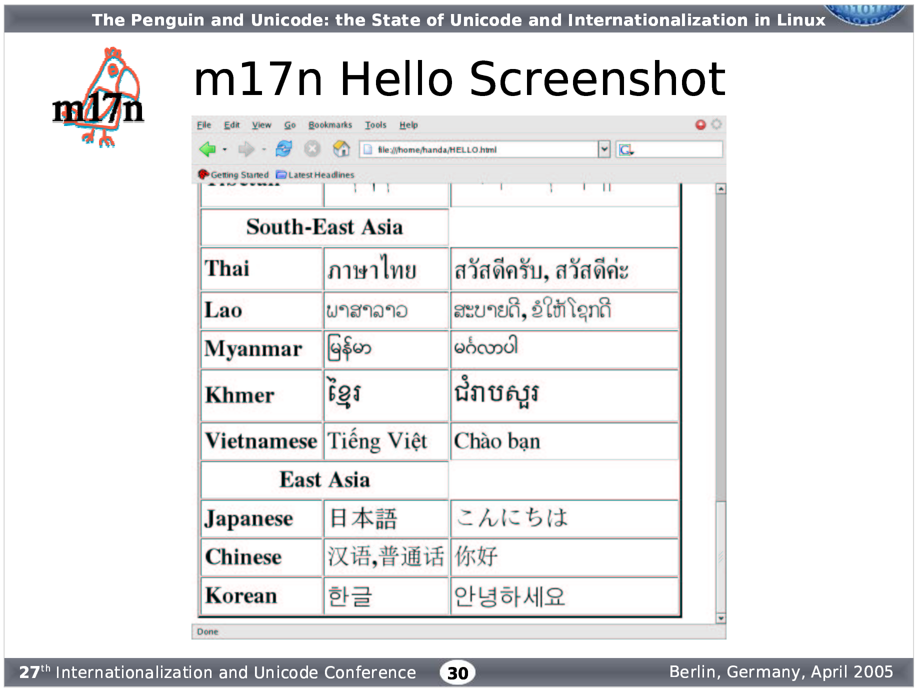
Bengali	☆
Gujarati	☆
Gurmukhi	☆
Devanagari	☆
Kannada	☆
Malayalam	☆
Oriya	☆
Sinhala	☆
Tamil	☆
Telugu	☆

27th Internationalization and Unicode Conference 29 Berlin, Germany, April 2005

The m17n team appears to be hard at work insuring that all scripts defined in Unicode are supported. Notice that m17n has support for both Khmer and Myanmar, as well as Thaana and Sinhala. I could not find Syriac listed on the m17n web site: I wonder if this is just an omission.

The m17n web site provides a demo where you can submit a sample of text to be rendered on their server. This is a good way to prove to yourself how well a script is supported. (My only disappointment using the web demo was that the web application does not let you choose fonts, as some of the default fonts used by the web app are not good. For example, Thai is rendered using the Bitstream Cyberbit font which has typographically incorrect placement of certain vowels and tone marks --perhaps they should take a look at my **Unicode Font Guide for Free/Libre Open Source Operating Systems**[1]).

1. http://eyegene.ophthy.med.umich.edu/unicode/fontguide/.



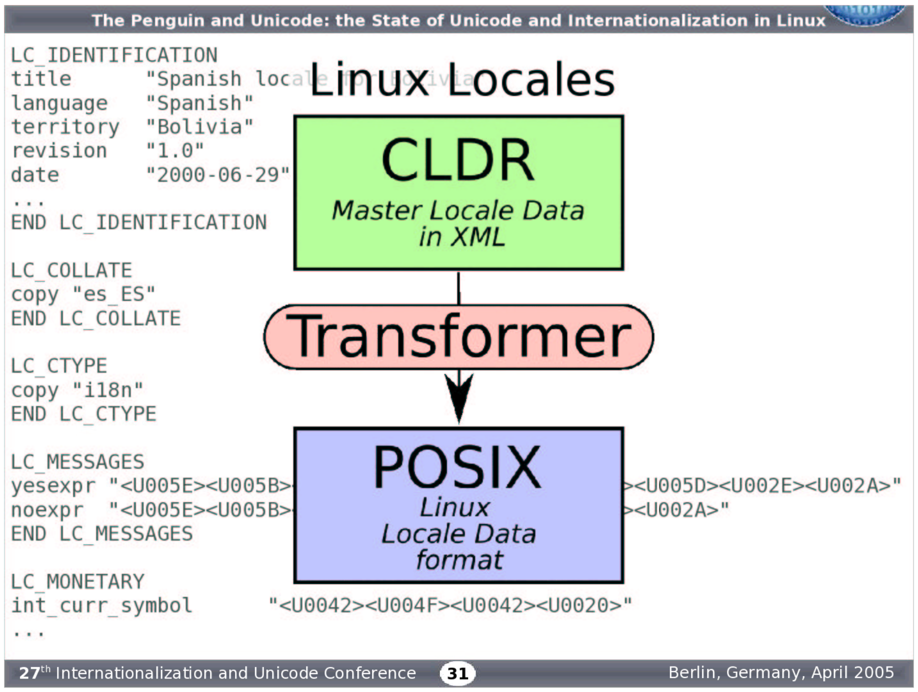
m17n Hello Screenshot

File Edit View Go Bookmarks Tools Help
file:///home/handa/HELLO.html
Getting Started Latest Headlines
Done

South-East Asia		
Thai	ภาษาไทย	สวัสดีครับ, สวัสดีค่ะ
Lao	ພາສາລາວ	ສະບາຍດີ, ຂໍໃຫ້ໂຊກດີ
Myanmar	မြန်မာ	မင်္ဂလာပါ
Khmer	ខ្មែរ	ជំរាបសួរ
Vietnamese	Tiếng Việt	Chào bạn
East Asia		
Japanese	日本語	こんにちは
Chinese	汉语,普通话	你好
Korean	한글	안녕하세요

Here is a screenshot of the Firefox browser from Mozilla.org compiled to use the m17n library, shown here rendering “Hello” in Southeast Asian and East Asian languages. This is one of the m17n team's experimental projects. (They have also tied m17n to the Cairo[1] SVG rendering library and to the MagicPoint[2] presentation program). Notice the Myanmar and Khmer examples. (In this screen shot you can also see the problem with using Bitstream Cyberbit for rendering Thai).

1. Cairo Vector Graphics Library, http://www.cairographics.org/.
2. MagicPoint Presentation Software, http://member.wide.ad.jp/wg/mgp/.



Linux Locales

```
LC_IDENTIFICATION
title       "Spanish loca
language    "Spanish"
territory   "Bolivia"
revision    "1.0"
date        "2000-06-29"
...
END LC_IDENTIFICATION

LC_COLLATE
copy "es_ES"
END LC_COLLATE

LC_CTYPE
copy "i18n"
END LC_CTYPE

LC_MESSAGES
yesexpr  "<U005E><U005B>                          ><U005D><U002E><U002A>"
noexpr   "<U005E><U005B>                          ><U002A>"
END LC_MESSAGES

LC_MONETARY
int_curr_symbol       "<U0042><U004F><U0042><U0020>"
...
```

So far we have covered message translation and internationalized text display in Linux. To complete the internationalization equation, let's look at locale data. Locale data are important for insuring the proper display of sorted lists (i.e., collation), monetary currencies, numbers, times, calendar dates, and other regional preferences.

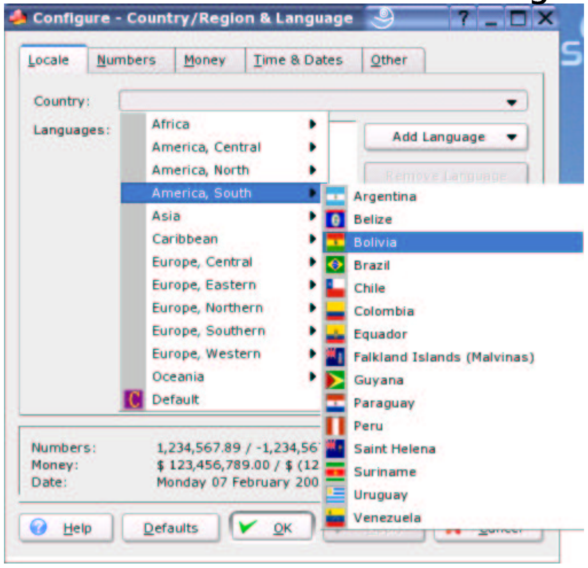
Glibc is the GNU C library implementation for GNU Hurd[1] and Linux systems. In addition to providing a high-performance and standards-conformant[2] C library, it is extensively internationalized. The latest release, version 2.3.4, is packaged with 189 locales. The locale data used by glibc are in the POSIX format, a sample of which is shown in the background here.

The Unicode Consortium's Common Locale Data Repository (CLDR)[3] project was originally developed under the sponsorship of the Linux Application Development Environment (LADE) of the Free Standards Group[4]'s Open Internationalization Initiative[5]. Locale data in CLDR are collected and stored in an XML format known as the Locale Data Markup Language (LDML)[6]. One goal of the CLDR was to be compatible with existing locale specifications, such as the POSIX specification. A transformer is used to translate locale data in the master XML format into the POSIX format required by glibc[7]. Note that CLDR is not just for Linux – it is the master locale data repository for IBM's ICU framework and for Java as well.

1. GNU Hurd: http://www.gnu.org/software/hurd/hurd.html.
2. GNU glibc conforms to ISO C 99, POSIX.1c, POSIX.1j, POSIX.1d, Unix98 and the Single Unix Specification. See http://www.gnu.org/software/libc/libc.html.
3. Unicode Common Locale Data Repository: http://www.unicode.org/press/pr-cldr1.2.html.
4. Free Standards Group, http://www.freestandards.org.
5. Open Internationalization Initiative, http://www.openi18n.org.
6. Mark Davis, **LDML**, Unicode Technical Standard #35, http://www.unicode.org/reports/tr35/.
7. Kentaro Noji and Tetsuji Orita, **Open Source Project for Unicode Locales**, http://www.unicode.org/iuc/iuc18/papers/a2-paper.pdf.

The Penguin and Unicode: the State of Unicode and Internationalization in Linux

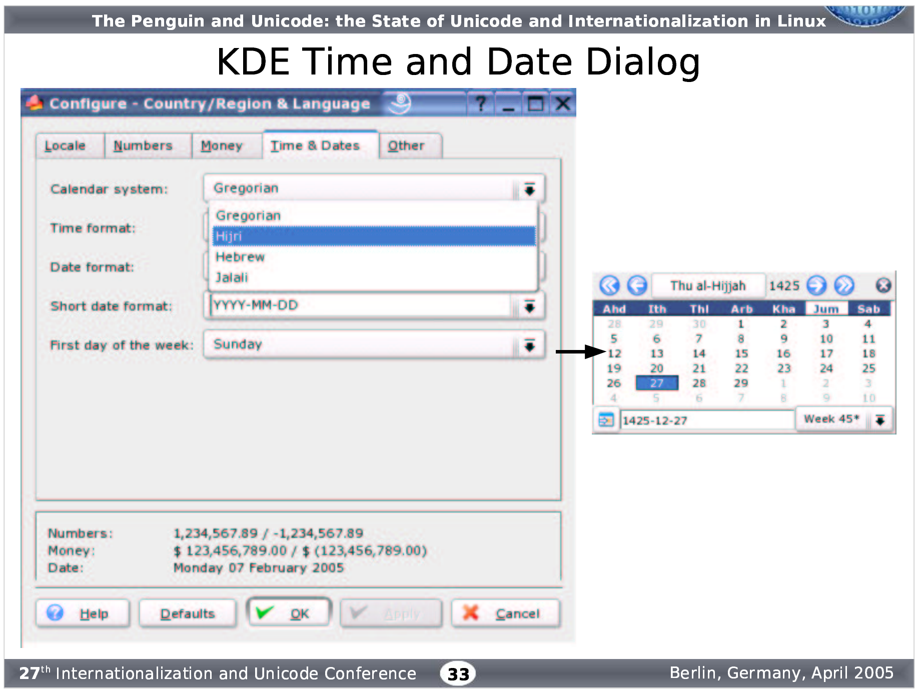
KDE Locale Selection Dialog



Numbers: 1,234,567.89 / -1,234,56
Money: $ 123,456,789.00 / $ (12
Date: Monday 07 February 200

27th Internationalization and Unicode Conference 32 Berlin, Germany, April 2005

For the end user, changing or modifying the locale in the modern desktop environment provided by KDE is quite simple. The KDE Country, Region and Language Dialog shown here may be accessed from the KDE Control Center. Individual preferences for numbers, dates, monetary and calendar display can be specified to override the defaults provided for a given locale. Changes affect newly-started applications immediately.



KDE Time and Date Dialog

Calendar system: Gregorian

Gregorian
Hijri
Hebrew
Jalali

Time format:

Date format:

Short date format: YYYY-MM-DD

First day of the week: Sunday

Thu al-Hijjah 1425

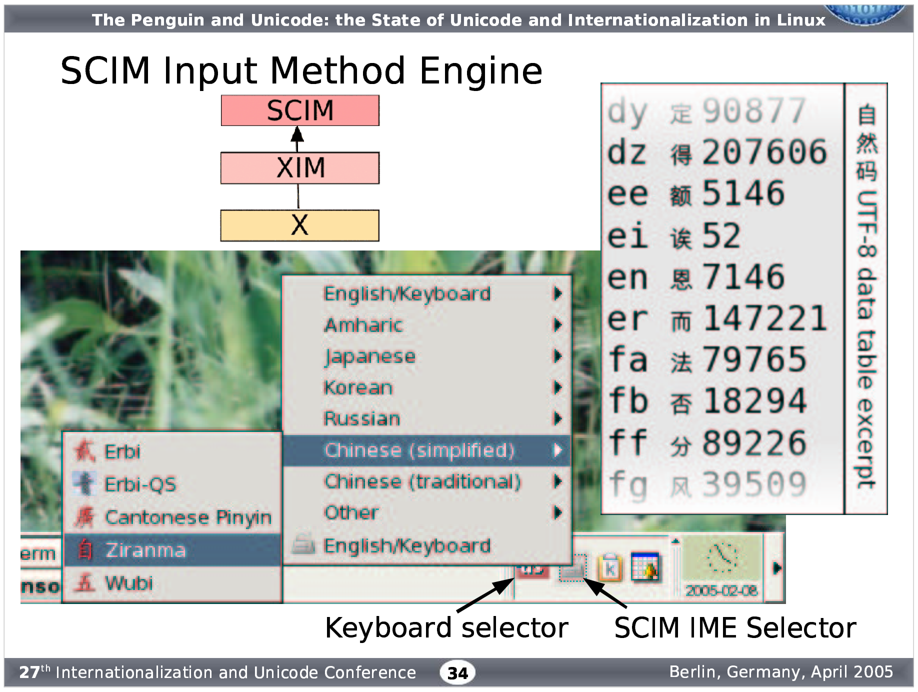
Ahd	Ith	Thl	Arb	Kha	Jum	Sab
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	1	2	3
4	5	6	7	8	9	10

1425-12-27 Week 45*

Numbers:	1,234,567.89 / -1,234,567.89
Money:	$ 123,456,789.00 / $ (123,456,789.00)
Date:	Monday 07 February 2005

In this screen shot, the Hijri calendar is being selected for use in an American English (i.e., “en_US.UTF-8”) locale.

Let's compare this feature with Windows XP. Windows XP also lets you choose the Hijri calendar: however, to use it, you must first select an Arabic locale. KDE does not impose such a restriction. The fact that the Hijri calendar is not dependent on an Arabic locale in KDE is a natural outcome of the modular design of almost every facet of Linux.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

SCIM Input Method Engine

$$\begin{tikzcd} \boxed{\text{SCIM}} \\ \boxed{\text{XIM}} \arrow[u] \\ \boxed{\text{X}} \arrow[u, dash] \end{tikzcd}$$

dy	定	90877
dz	得	207606
ee	额	5146
ei	诶	52
en	恩	7146
er	而	147221
fa	法	79765
fb	否	18294
ff	分	89226
fg	风	39509

自然码 UTF-8 data table excerpt

Keyboard selector SCIM IME Selector

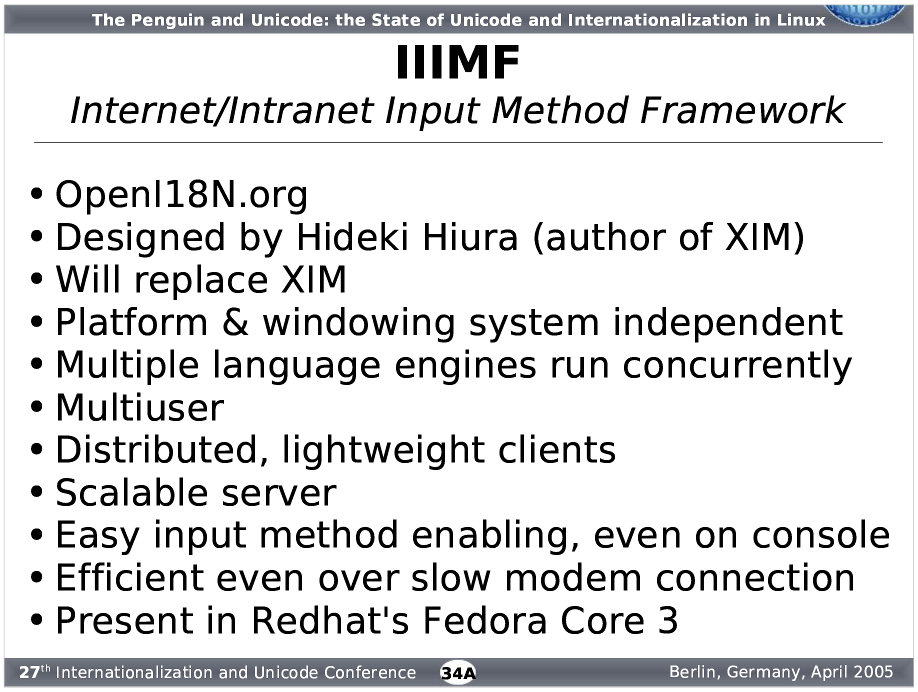
27th Internationalization and Unicode Conference 34 Berlin, Germany, April 2005

A final aspect of localization technology worth discussing is input method engines (IMEs). Input method software is an absolute necessity for languages such as Chinese, Japanese, and Korean. In East Asia, users typically enter word pronunciation phonetically, and then pick characters or character compounds from a pick list. Intelligent input methods strive to sort the pick lists based on which characters are used most frequently, based on the context of the text already entered, or a combination of both. Input methods are also useful for alphabetic scripts, for example to make it easier for users to enter diacritical marks, or to guarantee canonical character ordering.

While other IMEs for Linux do exist, James Su's Smart Common Input Method (SCIM) platform[1] is arguably the best and most comprehensive. SCIM not only provides a user-friendly platform for end users, but also makes it very easy for developers to add additional input methods. The platform installs seamlessly into the desktop tray on both the KDE and Gnome desktops. SCIM currently provides a wide range of input methods covering Chinese, Japanese, Korean, modern Vietnamese, Amharic, and Russian. Composing and dead key support is built in. On the backend, SCIM talks to XIM.

Compared to, say, trying to write or decipher an XKB keyboard map for X11, adding a new input method to SCIM is very easy to do. Input method data files are simple UTF-8 files – a snippet from the 自然码 *zìránmǎ* (自然双拼 *zìránshuāngpīn)* table is shown above to give you an idea*.* The simple UTF-8 format makes it easy to understand and debug. The first column indicates what keys on a QWERTY keyboard to type, the second column shows the Chinese character, and the third column contains a number related to that character's usage frequency. Input methods for syllabaries or alphabetic scripts do not require the third column.

1. SCIM: http://www.scim-im.org.



The Penguin and Unicode: the State of Unicode and Internationalization in Linux

IIIMF

Internet/Intranet Input Method Framework

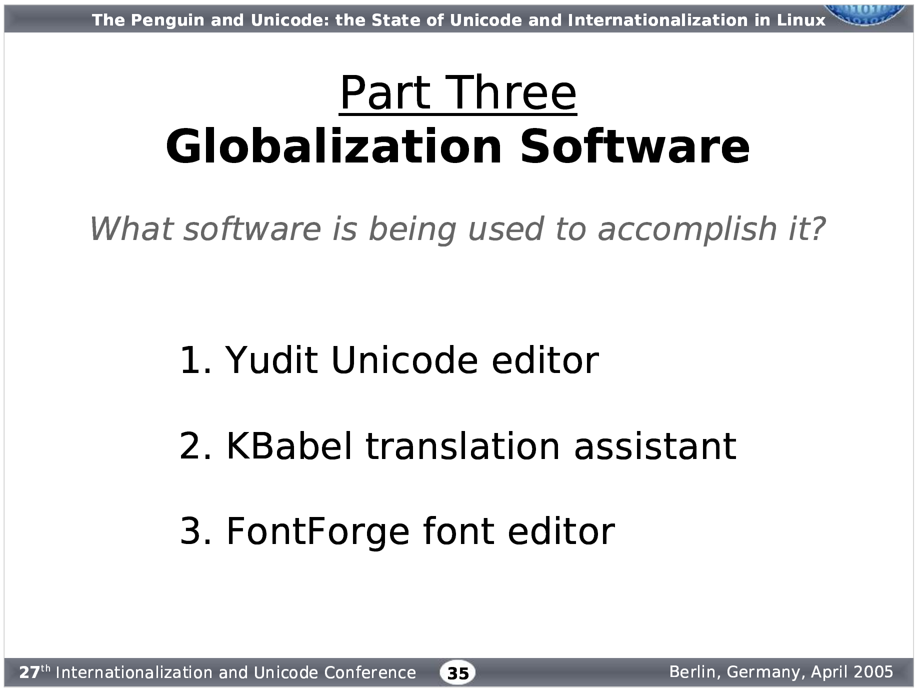
- OpenI18N.org
- Designed by Hideki Hiura (author of XIM)
- Will replace XIM
- Platform & windowing system independent
- Multiple language engines run concurrently
- Multiuser
- Distributed, lightweight clients
- Scalable server
- Easy input method enabling, even on console
- Efficient even over slow modem connection
- Present in Redhat's Fedora Core 3

27th Internationalization and Unicode Conference 34A Berlin, Germany, April 2005

Unlike the X Input Method framework which is tied to the X windowing architecture, IIIMF is a modern, platform and windowing system-independent input method framework developed by Hideki Hiura, the original author of XIM.

IIIMF is based on a modern client-server architecture which allows multiple language engines to run concurrently. IIIMF not only provides an X Window client framework (IIMXCF), it also provides client frameworks for Java2, Emacs, GTK+, QT and Windows in addition to the generic libiiimcf C client framework library.

You can currently try out IIIMF in Redhat's Fedora Core 3. Other Linux distributors are certain to follow suit in the near future.



Part Three
Globalization Software

What software is being used to accomplish it?

1. Yudit Unicode editor
2. KBabel translation assistant
3. FontForge font editor

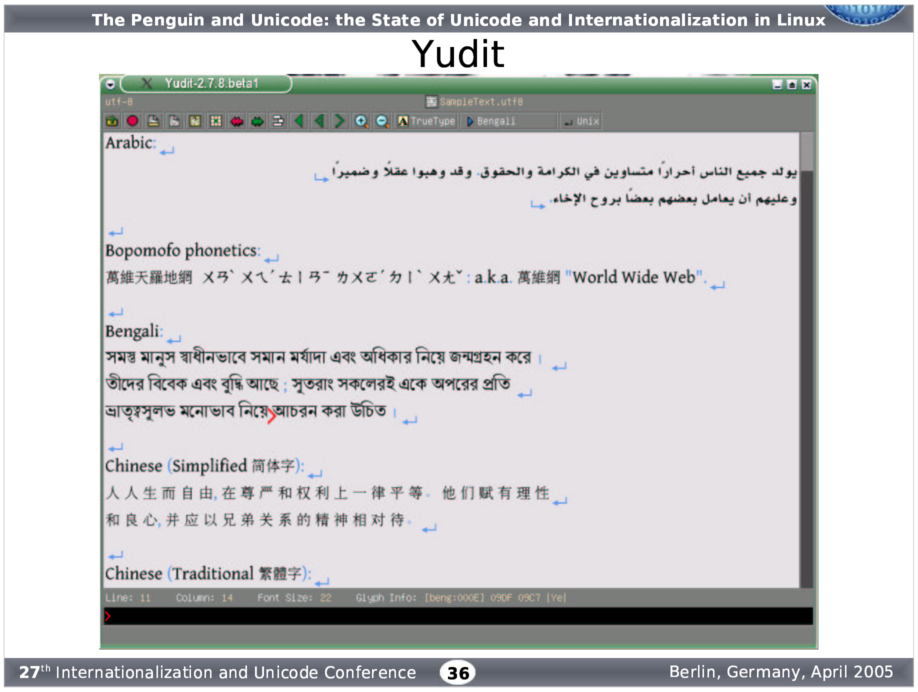
We have now completed our survey of infrastructure technologies in Linux.

In the final part of this talk, we are going to take a look at three Open Source productivity applications which have played --and continue to play-- key roles in the Linux globalization process. Although it is difficult to measure the impact that individual software packages have on a large phenomenon like the world-wide globalization of Linux, it is clear that these three have had significant contributing roles in the process.

1. **Yudit**, a Unicode editor

2. **KBabel**, a translation assistant, and

3. **Fontforge**, a font editor


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

Yudit

```
Arabic:
يولد جميع الناس أحراراً متساوين في الكرامة والحقوق. وقد وهبوا عقلاً وضميراً
وعليهم أن يعامل بعضهم بعضاً بروح الإخاء.

Bopomofo phonetics:
萬維天羅地網 ㄨㄢˋ ㄨㄟˊ ㄊㄧㄢˉ ㄌㄨㄛˊ ㄉㄧˋ ㄨㄤˇ : a.k.a. 萬維網 "World Wide Web".

Bengali:
সমস্ত মানুস স্বাধীনভাবে সমান মর্যাদা এবং অধিকার নিয়ে জন্মগ্রহন করে ।
তাঁদের বিবেক এবং বুদ্ধি আছে ; সুতরাং সকলেরই একে অপরের প্রতি
ভ্রাতৃত্বসুলভ মনোভাব নিয়ে আচরন করা উচিত ।

Chinese (Simplified 简体字):
人 人 生 而 自 由, 在 尊 严 和 权 利 上 一 律 平 等。 他 们 赋 有 理 性
和 良 心, 并 应 以 兄 弟 关 系 的 精 神 相 对 待。

Chinese (Traditional 繁體字):
Line: 11    Column: 14    Font Size: 22    Glyph Info: [beng:000E] 09DF 09C7 |Ye|
```

27th Internationalization and Unicode Conference 36 Berlin, Germany, April 2005

Yudit[1] is an Open Source Unicode text editor written by Gaspar Sinai. Gaspar had wanted to be able to write Hungarian and Japanese in a single document. He realized that without much additional work, he could create a general Unicode text editor. Yudit can be used under any locale setting. It can even be used on the various free BSD operating systems, like OpenBSD which currently lacks an NLS implementation (A group of Japanese developers has started the Citrus[2] project to create a complete POSIX NLS implementation with locale/iconv and a non-GPL gettext for the *BSD OSes, but I don't think anything is actually available yet).

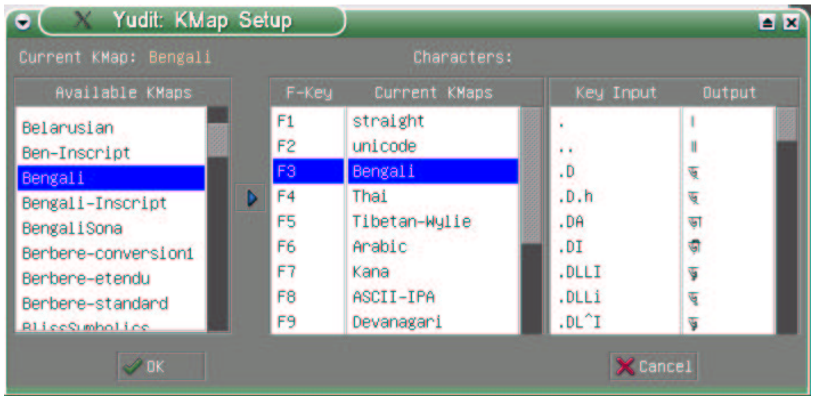
Yudit is easy to compile (only dependent on X11 libraries), extremely easy to use, and comes with a huge set of keyboard maps. The program even comes with handwriting recognition for Chinese Hanzi and Japanese Kanji (This is a very nice touch, although in actual practice it is much faster to use an external input method engine for CJK text).

In addition, the distribution includes two indispensible utilities: uniprint, which provides Postscript-based printing of your Unicode text, and uniconv, which seamlessly converts files between a large number of Unicode and legacy ISO and national encoding formats.

Yudit is now also available for use on Windows too.

1. Yudit, http://www.yudit.org.
2. Citrus Project, http://people.freebsd.org/~imp/index-en.html.

Yudit Keymap Selection Dialog



Current KMap: Bengali

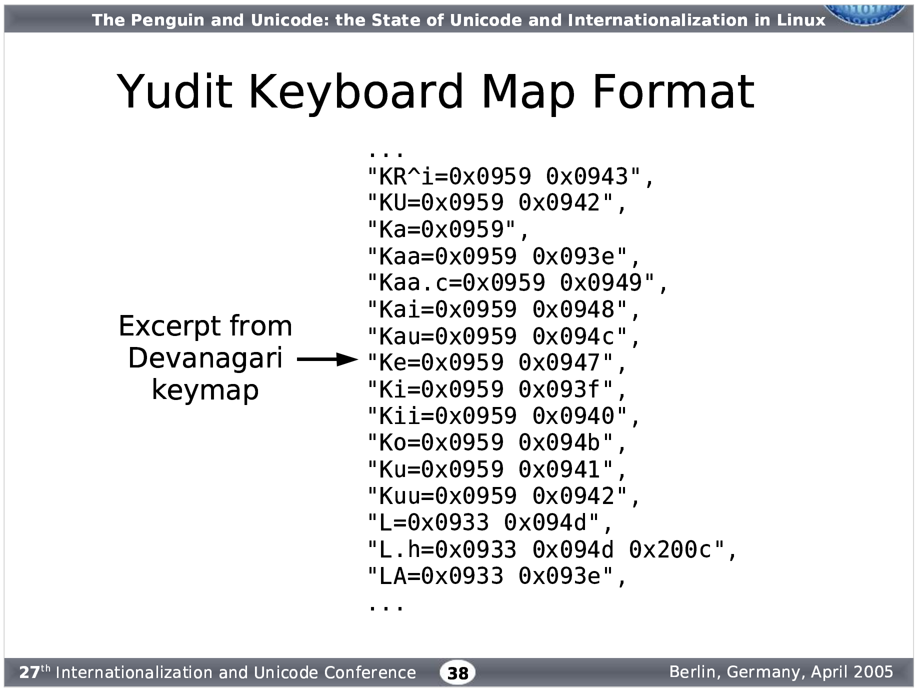
Characters:

Available KMaps
Belarusian
Ben-Inscript
Bengali
Bengali-Inscript
BengaliSona
Berbere-conversion1
Berbere-etendu
Berbere-standard
BlissSymbolics

F-Key	Current KMaps
F1	straight
F2	unicode
F3	Bengali
F4	Thai
F5	Tibetan-Wylie
F6	Arabic
F7	Kana
F8	ASCII-IPA
F9	Devanagari

Key Input	Output
.	।
..	॥
.D	ড
.D.h	ঢ
.DA	ডা
.DI	ডী
.DLLI	ডৣ
.DLLi	ডৢ
.DL^I	ডৣ

This slide shows Yudit's keyboard map selection dialog. The current beta version of Yudit, v. 2.7.8, comes with an astounding number of keyboard maps – 125 in all – which have been contributed by Yudit users from around the world.

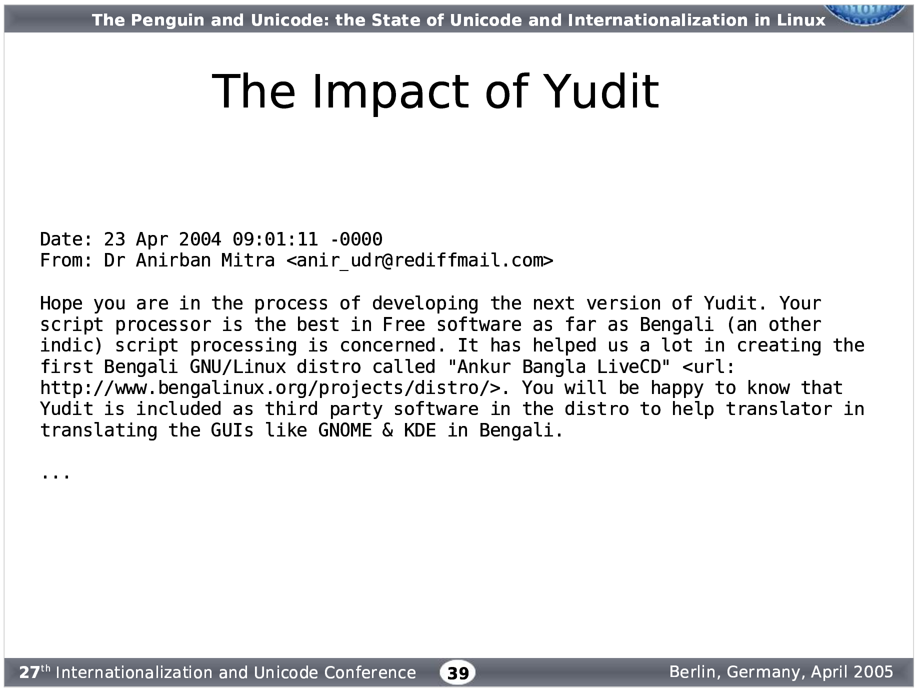


Yudit Keyboard Map Format

Excerpt from Devanagari keymap →

```
...
"KR^i=0x0959 0x0943",
"KU=0x0959 0x0942",
"Ka=0x0959",
"Kaa=0x0959 0x093e",
"Kaa.c=0x0959 0x0949",
"Kai=0x0959 0x0948",
"Kau=0x0959 0x094c",
"Ke=0x0959 0x0947",
"Ki=0x0959 0x093f",
"Kii=0x0959 0x0940",
"Ko=0x0959 0x094b",
"Ku=0x0959 0x0941",
"Kuu=0x0959 0x0942",
"L=0x0933 0x094d",
"L.h=0x0933 0x094d 0x200c",
"LA=0x0933 0x093e",
...
```

The uncompiled keyboard maps are in a simple plain text format. Here's an excerpt from a phonetically-based Devanagari keymap for Yudit. As you can see, any number of keystrokes from a plain QWERTY keyboard can be mapped to any sequence of Unicode values. When the keyboard is used to type Devanagari, Yudit will of course apply the correct shaping and glyph reordering rules needed for the display of the Devanagari script.



The Penguin and Unicode: the State of Unicode and Internationalization in Linux

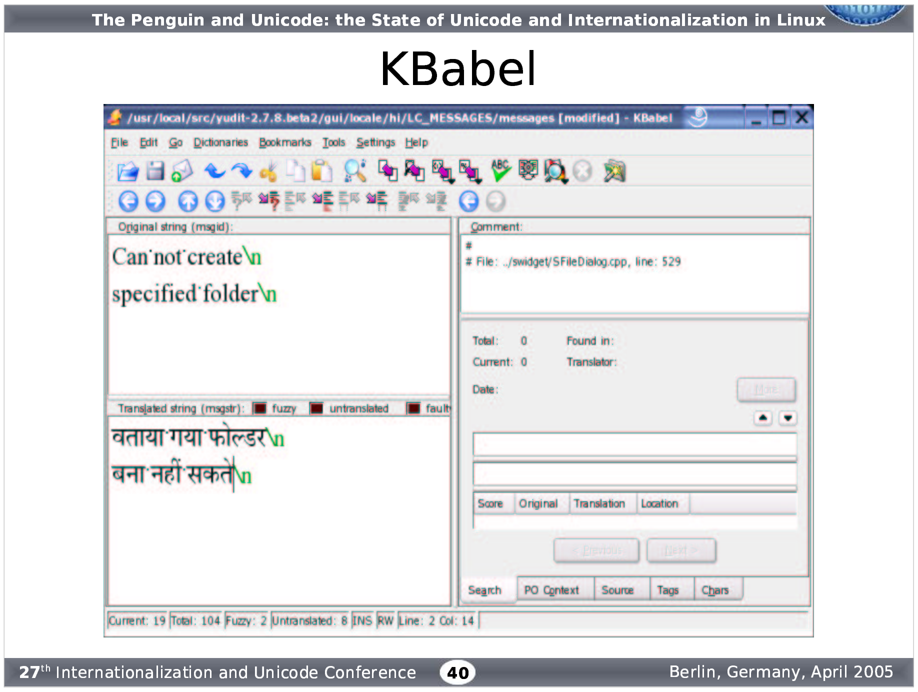
The Impact of Yudit

```
Date: 23 Apr 2004 09:01:11 -0000
From: Dr Anirban Mitra <anir_udr@rediffmail.com>

Hope you are in the process of developing the next version of Yudit. Your
script processor is the best in Free software as far as Bengali (an other
indic) script processing is concerned. It has helped us a lot in creating the
first Bengali GNU/Linux distro called "Ankur Bangla LiveCD" <url:
http://www.bengalinux.org/projects/distro/>. You will be happy to know that
Yudit is included as third party software in the distro to help translator in
translating the GUIs like GNOME & KDE in Bengali.

...
```

27th Internationalization and Unicode Conference 39 Berlin, Germany, April 2005

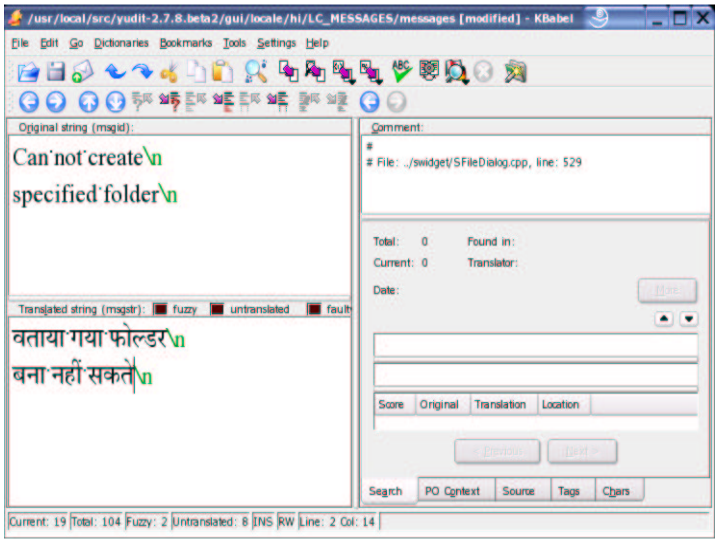
Because Yudit is simple to set up and simple to use on almost any computer, Yudit has had a significant impact in allowing localization teams the world over to work efficiently. I found the snippet of email shown above in part of the TODO.TXT file included with Yudit 2.7.8 beta 1. After complementing Gaspar Sinai on his software, Dr. Mitra goes on to talk about the Bengali Khanda Ta problem and make some suggestions for improving Yudit.

The open dialog between developers and end-users in Open Source projects, as illustrated in this example from Yudit, fosters a climate of rapid development and bug resolution.



The Penguin and Unicode: the State of Unicode and Internationalization in Linux

KBabel


/usr/local/src/yudit-2.7.8.beta2/gui/locale/hi/LC_MESSAGES/messages [modified] - KBabel
File Edit Go Dictionaries Bookmarks Tools Settings Help
Original string (msgid):
Translated string (msgstr): fuzzy untranslated faulty
Comment:
Total: 0
Current: 0
Found in:
Translator:
Date:
More
Score Original Translation Location
< Previous
Next >
Search PO Context Source Tags Chars

```
Can not create\n
specified folder\n
```

```
वताया गया फोल्डर\n
बना नहीं सकते\n
```

```
#
# File: ../swidget/SFileDialog.cpp, line: 529
```

```
Current: 19 Total: 104 Fuzzy: 2 Untranslated: 8 INS RW Line: 2 Col: 14
```

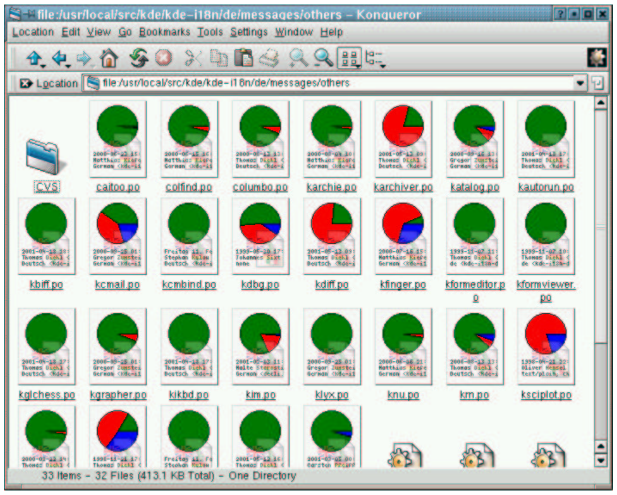
27th Internationalization and Unicode Conference 40 Berlin, Germany, April 2005

Matthias Keifer's KBabel[1] is a translation assistance tool for editing and managing GNU gettext PO files. The main PO file editor has the ability to search for translations in multiple dictionaries, includes spell and syntax checking, and has the ability to show diffs. The program also includes a catalog manager to help one keep track of PO files, a stand-alone dictionary application, and a “rough translation” tool which allows you to create a set of rough translations from dictionaries as a starting point.

1. KBabel, http://i18n.kde.org/tools/kbabel/.

The Penguin and Unicode: the State of Unicode and Internationalization in Linux

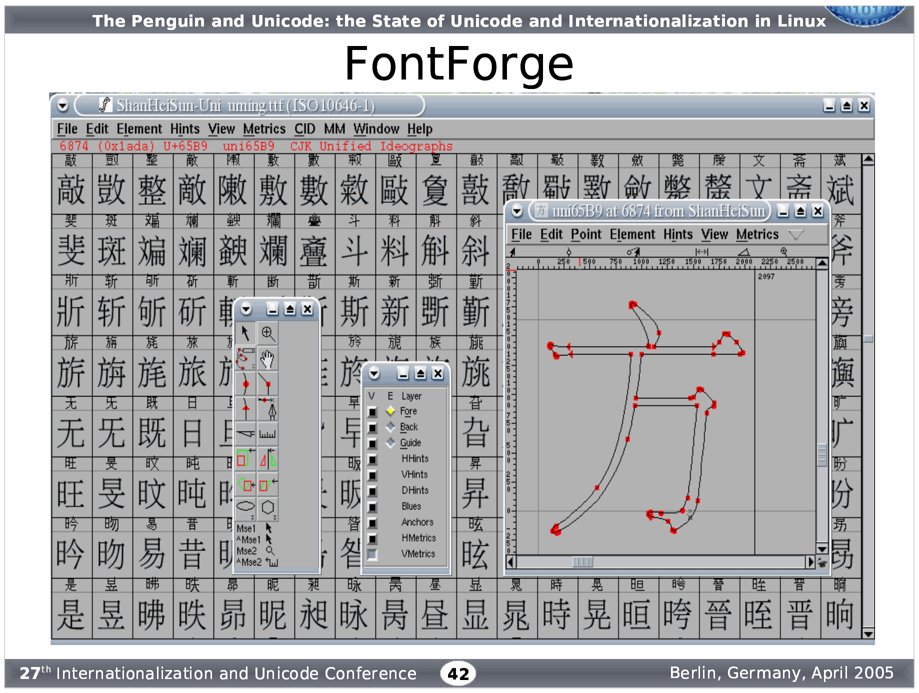
KBabel - Konqueror Integration


file:/usr/local/src/kde/kde-i18n/de/messages/others – Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location file:/usr/local/src/kde/kde-i18n/de/messages/others
CVS
caitoo.po
colfind.po
columbo.po
karchie.po
karchiver.po
katalog.po
kautorun.po
kbiff.po
kcmail.po
kcmbind.po
kdbg.po
kdiff.po
kfinger.po
kformeditor.po
kformviewer.po
kglchess.po
kgrapher.po
kikbd.po
kim.po
klyx.po
knu.po
krn.po
ksciplot.po
33 Items - 32 Files (413.1 KB Total) - One Directory

27th Internationalization and Unicode Conference 41 Berlin, Germany, April 2005

KBabel is well-integrated into the KDE desktop. Konqueror is the combined file manager and web browser of the KDE desktop. In this screen shot, Konqueror is being used to browse the contents of a directory containing PO files. KBabel's summary statistics regarding the translation status of each PO file are being displayed on-the-fly in this preview mode. In each pie chart, green indicates translated messages, red untranslated, and blue “fuzzy”.

While many people just use Yudit or some other editor to create PO translation files, I think you can see that KBabel provides a set of convenience and management features which can streamline the tedious process of localizing software, especially in large software projects like KDE.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

FontForge

ShanHeiSun-Uni uming.ttf (ISO10646-1)
File Edit Element Hints View Metrics CID MM Window Help
6874 (0x1ada) U+65B9 uni65B9 CJK Unified Ideographs
uni65B9 at 6874 from ShanHeiSun
File Edit Point Element Hints View Metrics
Layer
Fore
Back
Guide
HHints
VHints
DHints
Blues
Anchors
HMetrics
VMetrics

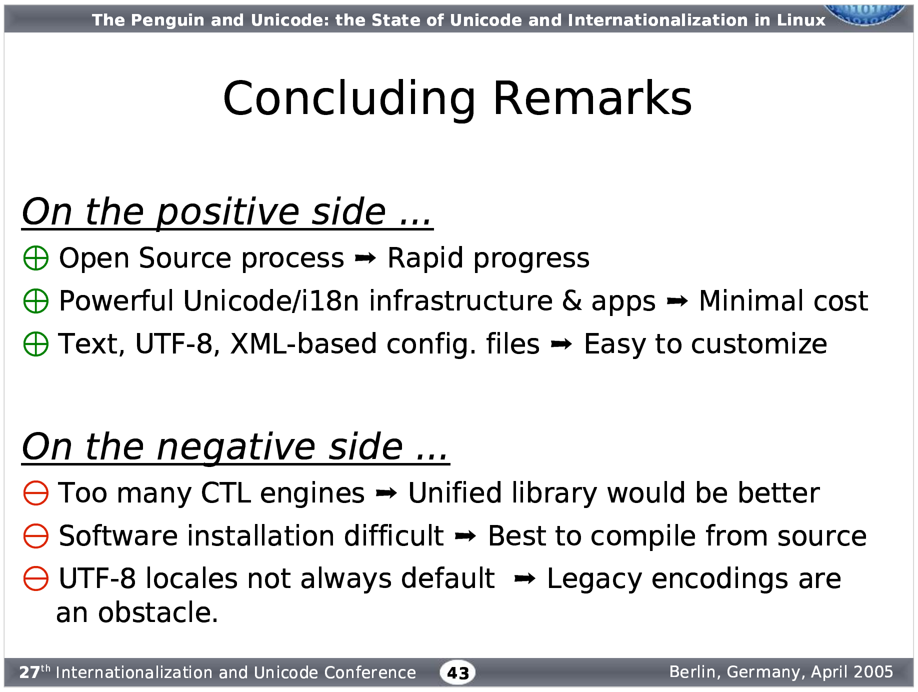
27th Internationalization and Unicode Conference 42 Berlin, Germany, April 2005

FontForge[1] is an outline font editor written by George Williams. FontForge can be used to create or edit Postscript, TrueType, OpenType, CID-keyed, multi-master, CFF, SVG and BDF fonts. The program will also permit you to convert fonts from one format to another.

Here the Arphic Unicode Ming font is shown being edited. Arphic Technologies of Taiwan generously released a set of Big5 and GB2312 fonts under an Open Source license. Arne Goetje of the Debian project in Taiwan is now using FontForge to create Unicode-encoded fonts for use on Linux based on the glyphs from the legacy Arphic fonts[2]. He is also adding new glyphs for the Bopomofo extensions for Hakka and Minnan added to Unicode 4.0, as well as Han characters in the Hong Kong Supplemental Character Set (HKSCS).

The Taiwan Debian CJK font project is just one example of many Open Source font development projects made possible by the existance of this free/libre font editor.

1. Fontforge, http://fontforge.sourceforge.net/.
2. Arphic CJK fonts to Unicode project, http://debian.linux.org.tw/pub/3Anoppix/people/arne/.



Concluding Remarks

On the positive side ...

⊕ Open Source process ➡ Rapid progress

⊕ Powerful Unicode/i18n infrastructure & apps ➡ Minimal cost

⊕ Text, UTF-8, XML-based config. files ➡ Easy to customize

On the negative side ...

⊖ Too many CTL engines ➡ Unified library would be better

⊖ Software installation difficult ➡ Best to compile from source

⊖ UTF-8 locales not always default ➡ Legacy encodings are an obstacle.

Linux is not a static system: it is a system that is changing and maturing rapidly. The state of Unicode and internationalization in Linux today reflects this condition of rapid growth in both positive and negative ways.

On the positive side:

① The Open Source development process creates an open dialog between developers and users which results in rapid progress. We saw an example of this with the Unicode editor Yudit which facilitated the efforts of the Bengali Linux localization project.

② A solid Unicode-based infrastructure and powerful applications are available to both developers and end users at minimal cost or nearly free. A professional font design program for Windows or Mac costs about U.S. $ 349.00, or nearly 1/5 the annual salary of a person in Bangladesh. In contrast, George Williams' **FontForge** program is free and yet has a sufficiently powerful feature set for professional work.

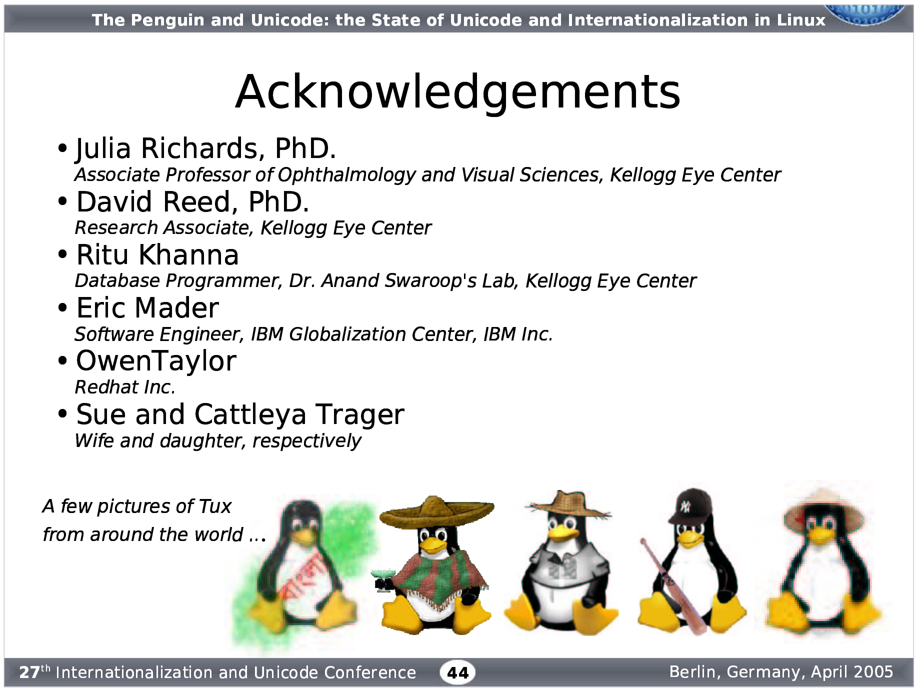
③ Essentially all configuration files, locale files, message translation files, keyboard maps, and input method maps are in plain ASCII text, plain UTF-8 text, or XML formats. This makes Linux systems easy to customize using a simple text editor like **vi** or **Yudit**.

On the negative side:

① There are too many complex text layout engines. It would be better if efforts were unified in the design of a single, toolkit-independent library following the examples provided by FontConfig and FreeType. Will the “FreeType Layout” library (not released yet) emerge as a future contender for the CTL title?

② Software installation on Linux is still too difficult for the average user. In real life, the RPM software package system used by Redhat and SuSE doesn't work as well as some competing systems (Debian's apt-get, for example). Compiling from source works best, but some “user” distributions don't have developer tools installed by default.

③ Although Redhat and SuSE now use UTF-8 locales by default, many of the other Linux distributions still default to legacy encodings. This is an unecessary obstacle.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

Acknowledgements

- Julia Richards, PhD.
  Associate Professor of Ophthalmology and Visual Sciences, Kellogg Eye Center
- David Reed, PhD.
  Research Associate, Kellogg Eye Center
- Ritu Khanna
  Database Programmer, Dr. Anand Swaroop's Lab, Kellogg Eye Center
- Eric Mader
  Software Engineer, IBM Globalization Center, IBM Inc.
- OwenTaylor
  Redhat Inc.
- Sue and Cattleya Trager
  Wife and daughter, respectively

A few pictures of Tux from around the world ...

27th Internationalization and Unicode Conference 44 Berlin, Germany, April 2005

This concludes my talk.

I'd like to thank:

- Julia Richards, PhD., for providing financial support and allowing me to spend some of my time on this project.
- My colleague David Reed, for reading over the draft and providing valuable feedback.
- My colleague Ritu Khanna, for help in evaluating and testing numerous Unicode-related things, and especially for her help with Indic script support issues.
- Eric Mader of IBM and Owen Taylor of Redhat for answering my questions about complex text layout in ICU and Pango, respectively.
- My wife Sue and daughter Cattleya, for their loving support and helpful suggestions.

The Penguin and Unicode: the State of Unicode and Internationalization in Linux

27th Internationalization and Unicode Conference 45 Berlin, Germany, April 2005

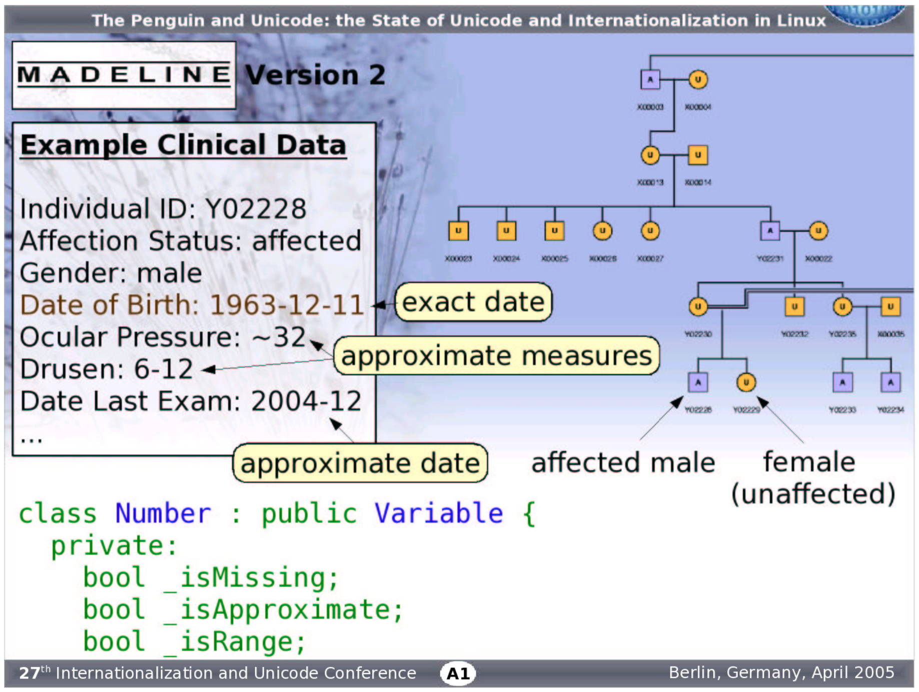
About the Presenter

Ed Trager is a bioinformatics programmer at the Kellogg Eye Center at the University of Michigan in Ann Arbor, USA. He is the author of **Madeline**, a program used by researchers conducting "gene scan" linkage studies to search for genes responsible for inherited human diseases, and the lead architect of **Gladiator**, a web-based clinical and genetics data entry system for use in research hospitals[1].

In addition to his interests in bioinformatics, Ed has always been interested in human languages and cultures. He has spent years in Asia where he studied Chinese and Thai. Ed's computational and linguistic pursuits have provided fertile ground in which his interest in Unicode and software internationalization has sprouted.

He wrote and maintains the web-based guide, ***A Quick Primer On Unicode and Software Internationalization Under Linux and Related Free/Libre Open Source Operating Systems***[2], to which he recently added the resource, ***Unicode Font Guide For Free/Libre Open Source Operating Systems***[3].

1. See http://eyegene.ophthy.med.umich.edu for links to Madeline and Gladiator.
2. Unicode page: http://eyegene.ophthy.med.umich.edu/unicode/.
3. Font Guide: http://eyegene.ophthy.med.umich.edu/unicode/fontguide/.


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

MADELINE Version 2

Example Clinical Data

Individual ID: Y02228
Affection Status: affected
Gender: male
Date of Birth: 1963-12-11
Ocular Pressure: ~32
Drusen: 6-12
Date Last Exam: 2004-12
…

```
class Number : public Variable {
  private:
    bool _isMissing;
    bool _isApproximate;
    bool _isRange;
```

27th Internationalization and Unicode Conference A1 Berlin, Germany, April 2005

The slides in this appendix highlight a few aspects of the software development work that I am currently leading in Dr. Julia Richards lab at the Kellogg Eye Center, University of Michigan, Ann Arbor.

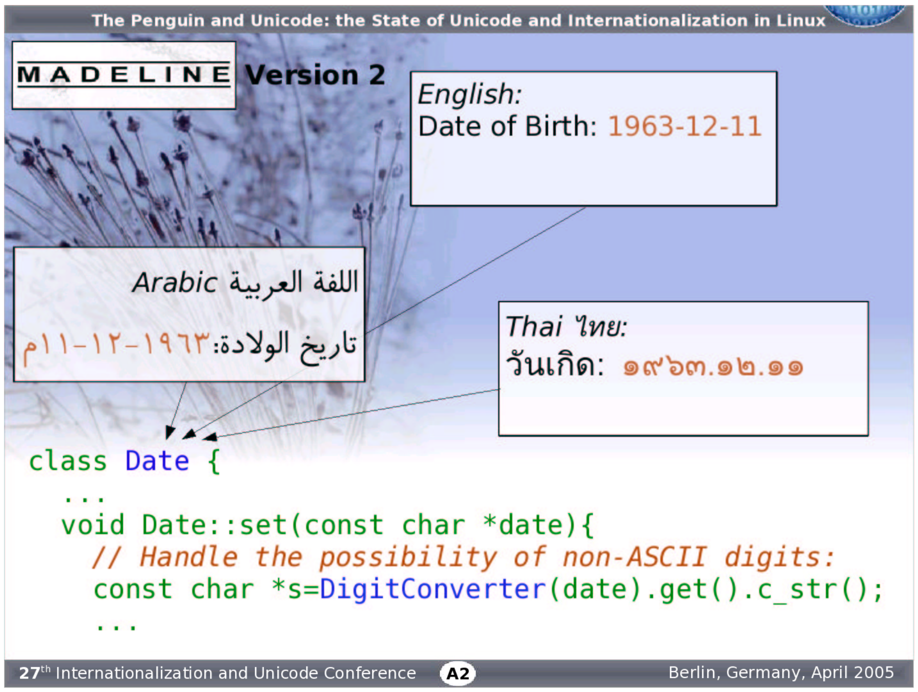
Madeline is a program I developed for facilitating data management in genetic linkage studies. Linkage analysis is a technique for finding the disease-causing "needles" (genes) in the genetic "haystack" of a family.

Madeline fundamentally consists of a database engine merged with an engine that understands how people are related to one another in pedigrees. In addition to being able transform data into numerous formats and draw pedigrees used in genetic linkage analysis, the program allows one to run queries that require an understanding of the relationships between people. For example, I might ask the program to show me all of the families that have a pair of affected siblings where both parents are unaffected.

This slide highlights some of the new things we are working on for Madeline version 2 which will be a complete re-write of the program from the ground up. We are taking an object-oriented approach in redesigning the program using modern C++.

In collecting medical information on extended families, it is common to obtain only approximate information about some family members or sampled individuals. Doctors may record approximate values in medical records, and patients may not be able to recall exactly when they began having symptoms.

Most database systems simply cannot handle approximations, but in Madeline 2 we are designing classes which will be able to distinguish and record precise values, missing values, and imprecise values in the form of approximations or ranges …


The Penguin and Unicode: the State of Unicode and Internationalization in Linux

MADELINE **Version 2**

English:
Date of Birth: 1963-12-11

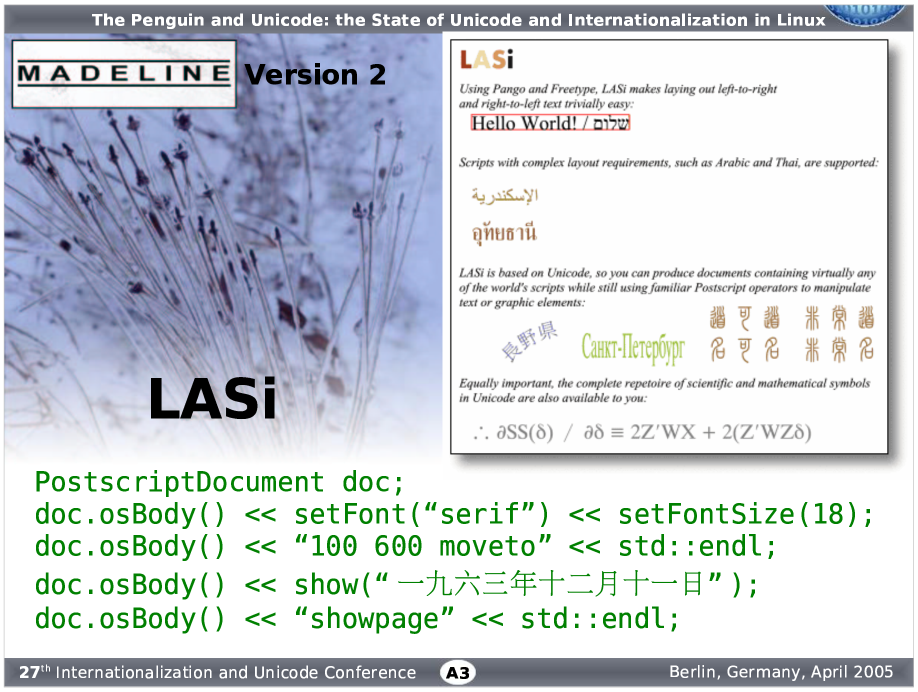
Arabic اللغة العربية
تاريخ الولادة:١٩٦٣-١٢-١١م

Thai ไทย:
วันเกิด: ๑๙๖๓.๑๒.๑๑

```
class Date {
  ...
  void Date::set(const char *date){
    // Handle the possibility of non-ASCII digits:
    const char *s=DigitConverter(date).get().c_str();
    ...
```

27th Internationalization and Unicode Conference A2 Berlin, Germany, April 2005

In addition to being able to handle approximate numeric and date values, we have also implemented a digit converter class that converts numerals written in other scripts to common Arabic-Indic digits. Thus, as shown here, UTF-8 strings containing dates of birth written in other scripts can be passed directly to the set() method or constructor of the Date class. The DigitConverter class first performs a low-cost check of whether the string contains UTF-8 versus plain ASCII. If the string contains only ASCII, nothing needs to be done and the DigitConverter simply returns the string unchanged, thus minimizing overhead while still handling Unicode data in a seamless manner.



The Penguin and Unicode: the State of Unicode and Internationalization in Linux

MADELINE Version 2

LASi

LASi

Using Pango and Freetype, LASi makes laying out left-to-right and right-to-left text trivially easy:

Hello World! / שלום

Scripts with complex layout requirements, such as Arabic and Thai, are supported:

الإسكندرية

อุทัยธานี

LASi is based on Unicode, so you can produce documents containing virtually any of the world's scripts while still using familiar Postscript operators to manipulate text or graphic elements:

長野県 Санкт-Петербург

Equally important, the complete repetoire of scientific and mathematical symbols in Unicode are also available to you:

$\therefore \partial SS(\delta) \;/\; \partial\delta \equiv 2Z'WX + 2(Z'WZ\delta)$

```
PostscriptDocument doc;
doc.osBody() << setFont(“serif”) << setFontSize(18);
doc.osBody() << “100 600 moveto” << std::endl;
doc.osBody() << show(“一九六三年十二月十一日”);
doc.osBody() << “showpage” << std::endl;
```

27th Internationalization and Unicode Conference A3 Berlin, Germany, April 2005

Madeline version 2 will also include the **LASi** library written by Larry Siden[1] in order to be able to produce Postscript output such as pedigree drawings and plots of analysis results containing Unicode strings.

As shown above, LASi provides a Postscript document class with stream operators that allow the user to create a Postscript document using familiar postscript commands like “moveto”. And, in order to handle Unicode in the most seamless manner possible, LASi implements methods like setFont(), setFontSize(), and show() which are patterned after the Postscript setfont, setfontsize, and show operators but manage UTF-8 Unicode strings directly.

On the backend, LASi uses Pango and FreeType in order to provide Complex Text Layout (CTL) services for laying out scripts like Arabic and Thai. As a result, programmers can easily produce very attractive Postscript output containing Unicode text with no more than the most basic understanding of the Postscript language. The programmer is freed from the complex details of glyph substitution and reordering, Postscript font dictionaries, or horrible things like CID fonts.

1. LASi http://eyegene.ophthy.med.umich.edu/lasi.