The Linux Cyrillic HOWTO

Table of Contents

Table of Contents

The Linux Cyrillic HOWTO

Alexander L. Belikoff, (`abel@bfr.co.il`), Berger Financial Research Ltd.

v4.0, 23 January 1998

This document describes how to set up your Linux box to typeset, view and print the documents in the Russian language.

1.Administrativia

2.Theoretical background

3.Preparing your environment

4.Editing text

5.Using Cyrillic with mail and news

6.Browsing the Cyrillic Web

7.Cyrillic wordprocessing

8.Printing and PostScript

9.Cyrillic in PostScript

10.Print setup

11.Localization and Internationalization

12.Staying compatible

13.Bibliography

14.Summary of the various useful resources

1.Administrativia

1.1 Introduction

This document covers the things you need to successfully work with information containing cyrillic text (mostly Russian) under Linux. Although this document assumes your using Linux as an operating system, most of information presented is equally applicable to many other Unix flavors. I shall try to keep the distinction as visible as possible.

There are a number of popular Linux distributions. As an example system I describe the RedHat 4.1 Linux (Vanderbildt) − the one I am personally using. Nevertheless, I shall try to highlight the differences, if they exist, in other popular distributions, such as Debian GNU/Linux and Slackware Linux.

Since such setup directly modifies and extends the Operating System, you should understand, what you are doing. Even though I tried to keep things as easy as possible, having some experience with a given piece of software is an advantage. I am not going to describe what the X Window System is or how to typeset the documents with TeX and LaTeX, or how to install printer in Linux. Those issues are covered in other documents.

For the same reason, in most cases I describe a system−wide setup, by default requiring *root* privileges. Still, if there is a possibility for user−level setup, I'll try to mention it.

NOTE: The X Window System, TeX and other Linux components are complex systems with a sofisticated configuration. If you do something wrong, you can not only fail with Russian setup, but to break the component as well, if not the entire system. This is not to scare you off, but merely to make you understand the seriousness of the process and be careful. Preliminary backup of the config files is **highly** recommended. Having a guru around is also advantageous.

1.2 Availability and feedback

This document is available at sunsite.unc.edu or tsx−11.mit.edu as a part of the *Linux Document Project*. Also, it may be available at various FTP sites containing Linux. Moreover, it may be included as a part of Linux distribution.

If you have any suggestions or corrections regarding this document, please, don't hesitate to contact me as abel@bfr.co.il. Any new and useful information about Cyrillic support in various Unices is *highly appreciated*. Remember, it will help the others.

1.3 Acknowledgments and copyrights

Many people helped me (and not only me) with valuable information and suggestions. Even more people contributed software to the public community. I am sorry if I forgot to mention somebody.

So, here they go:

- Bas V. de Bakker
- David Daves
- Serge Vakulenko
- Sergei O. Naoumov
- Winfried Truemper
- Ilya K. Orehov
- Michael Van Canneyt
- Alex Bogdanov
- ...and the countless helpful people from the relcom.fido.ru.unix and relcom.fido.ru.linux Usenet newsgroups.

2.Theoretical background

2.1 Characters and codesets

In order to understand and print characters of various languages, the system and software should be able to distinguish them from other characters. That is, each unique character must have a unique representation inside the operating system, or the particular software package. Such collection of all unique characters, that the system is able to represent at once, is called a *codeset*.

At the time of the most operating system's creation, nobody cared about software being multilingual. Therefore, the most popular codeset was (and actually is) an *ASCII* (American Standard Code for Information Interchange).

The *standard ASCII* (aka 7−bit ASCII) comprises 128 unique codes. Some of them ASCII defines as real printable characters, and some are so−called *control characters*, which had special meanings in the old communication protocols. Each element of the set is identified by an integer *character code* (0−127). The subset of printable characters represents those found on the typewriter's keyboard with some minor additions. Each character occupies 7 least significant bits of a byte, whereas the most significant one was used for control purposes (say, transmission control in old communication packages).

The 7−bit ASCII concept was extended by 8−bit ASCII (aka *extended ASCII*). In this codeset, the characters' codes' range is 0−255. The lower half (0−127) is pure ASCII, whereas the upper one contains 127 more characters. Since this codeset is backward compatible with the ASCII (character still occupies 8 bit, the codes correspond the old ASCII), this codeset gained wide popularity.

The 8−bit ASCII doesn't define the contents of the upper half of the codeset. Therefore the ISO organization took the responsibility of defining a family of standards known as *ISO 8859−X* family. It is a collection of 8−bit codesets, where the lower half of each codeset (characters with codes 0−127) matches the ASCII and the upper parts define characters for various languages. For example, the following codesets are defined:

- `8859−1` − Europe, Latin America (also known as *Latin 1*)
- `8859−2` − Eastern Europe
- `8859−5` − Cyrillic
- `8859−8` − Hebrew

In Latin 1, the upper half of the table defines various characters which are not part of the English alphabet, but are present in various european languages (german umlauts, french accentes etc).

Another popular extended ASCII implementation is so−called *IBM codepage* (named after some computer company, that developed this codeset for it's infamous personal computers). This one contains pseudo−graphic characters in the upper half.

Software, that doesn't make any assumptions about the 8−th bit of the ASCII data is called *8−bit clean*. Some older programs, designed with 7−bit ASCII in mind are not 8−bit clean and may work incorrectly with your extended ASCII data. Most of packages, however, are able to deal with the extended ASCII by default, or require some very basic setup. **NOTE:** before posting the question *"I did all setup right, but I cannot enter/view Cyrillic characters!"*, please consult the section shells for the notes on the program, you are using.

For information about making your software 8−bit clean, see section locale−programming .

Since on most systems character occupies 8 bits, there is no way to extend ASCII more and more. The way to implement new symbols in ASCII−based codesets is creation of other extended ASCII implementations. This is the way, the Cyrillic ASCII set is implemented.

We already mentioned *ISO 8859−5* standard as the one defining the Cyrillic codeset. But as it often happens to the standards, this one was developed without taking into account the real practices in the former USSR. Therefore, one thing that standard really achieved was another degree of confusion. I wouldn't say that *ISO 8859−5* is widely used anywhere.

Other standards for Cyrillic include the so−called *Alt* codeset and *Microsoft CP1251* codepage. The former one was developed by (who?) for MS−DOS quite a while ago. Back then, there was not very buzz yet about internetworking, so the intention was to make it as compatible as possible with the IBM standard. Therefore the Alt codeset is effectively the same IBM codepage, where all specific European characters in the upper half were replaced with the Cyrillic ones, leaving the pseudographic ones. Therefore, it didn't screw the text windowing facilities and provided Cyrillic characters as well. The *Alt* standard is still alive and extremely popular in MS−DOS.

Microsoft CP1251 codepage is just an attempt of Microsoft to come up with the new standard for Cyrillic codeset in Windows. As far as I know, it is not compatible with anything else (not very surprizing, huh?)

And finally there is *KOI8−R*. This one is also quite old, but it was designed wisely and nowadays the design points of it look really useful.

Again, it is compatible with ASCII, and the Cyrillic characters are located in the upper half. But the main design point of *KOI8−R* is that the Cyrillic characters' positions must correspond to the English characters with the same phonetics. Namely, if we set the eighth bit of the English character `'a'`, we'll get the Cyrillic `'a'`. This means that, given the Cyrillic text written in KOI8−R, we can strip the eighth bit of each character *and we still get a readable text, although written with English characters!* This is very important now, since there are many mailers on the Internet, that just strip the eighth bit silently, being sure that every single soul on the face of the Earth speaks English.

Not surprisingly, *KOI8−R* quickly became a de−facto standard for Cyrillic on the Internet. Andrew A. Chernov did a tremendous amount of work to make a standard in this area. He is an author of RFC 1489 (*"Registration of a Cyrillic Character Set"*).

These two standards differ only in positions of the cyrillic characters in the table (that is in cyrillic character codes).

The principal difference is that the Alt codeset is used by MS−DOS users only, whereas KOI8−R is used in Unix, as well as in MS−DOS (though in the latter KOI8−R is much less popular). Since we are doing the right thing (namely working in the Unix operating system), we shall focuse mostly on KOI8−R.

As for the ISO standard, it is more popular in Europe and the US as a standard for Cyrillic. The leader in Russia is definitely KOI8−R.

There are other standards, which are different from ASCII and much more flexible. *Unicode* is most known. However, they are not implemented as good as the basic ones in Unix in general and Linux in particular. Therefore, I am not describing them here.

3.Preparing your environment

Before we start customizing various parts of the system functionality, we have to set up a couple basic things. Most of tools described below assume that there are Cyrillic fonts available and a user is able to input Cyrillic characters. To make it true we have to configure the environment to provide both fonts and input facility for Cyrillic.

There are effectively two interface models supported by Linux. One is the text mode, and the other one is the graphic mode, provided by the X Window System. Both require different setup, which will be described below.

3.1 Text mode setup

Generally, the text mode setup is the easiest way to show and input Cyrillic characters. There is one significant complication, however: the text mode fonts and keyboard layout manipulations depend on terminal driver implementation. Therefore, there is no portable way to achieve the goal across different systems.

Right now, I describe the way to deal with the Linux console driver. Thus, if you have another system, don't expect it to work for you. Instead, consult your terminal driver manual. Nevertheless, send me any information you find, so I'll be able to include it in further versions of this document.

Linux Console

The Linux console driver is quite a flexible piece of software. It is capable of changing fonts as well as keyboard layouts. To achieve it, you'll need the kbd package. Both RedHat and Slackware install kbd as part of a system.

The kbd package contains keyboard control utilities as well as a big collection of fonts and keyboard layouts.

Cyrillic setup with **kbd** usually involves two things:

1. Screen font setup. This is performed by the `setfont` program. The fonts files are located in `/usr/lib/kbd/consolefonts`. **NOTE:** Never run the `setfont` program under X because it will hang your system. This is because it works with low−level video card calls which X doesn't like.
2. Load the appropriate keyboard layout with the `loadkeys` program.

NOTE: In RedHat 3.0.3, `/usr/bin/loadkeys` has too restrictive access permissions, namely 700 (`rwx------`). There are no reasons for that, since everyone may compile his own copy and execute it (the appropriate system calls are not root−only). Thus, just ask your sysadmin to set more reasonable permissions for it (for example, 755).

The following is an excerpt from my `cyrload` script, which sets up the Cyrillic mode for Linux console:

```
if [ notset.$DISPLAY != notset. ]; then
    echo "`basename $0`:  cannot run under X"
    exit
fi

loadkeys /usr/lib/kbd/keytables/ru.map
setfont /usr/lib/kbd/consolefonts/Cyr_a8x16
mapscrn /usr/lib/kbd/consoletrans/koi2alt
echo -ne "\033(K"              # the magic sequence
echo "Use the right Ctrl key to switch the mode..."
```

Let me explain it a bit. You load the appropriate keyboard mapping. Then you load a font corresponding to the *Alt* codeset. Then, in order to be able to display text in *KOI8−R* correctly, you load a *screen translation* *table*. What it does is a translation of *some* characters from the upper half of the codeset to the *Alt* encoding. The word 'some' is crucial here − not all characters get translated, therefore some of them, like IBM pseudographic characters get unmodified to the screen and display correctly, since they are compatible with the *Alt* codeset, as opposed to *KOI8−R*. To ensure this, run **mc** and pretend you are back to MS−DOS 3.3...

Finally, the magic sequence is important but I have no idea what on the Earth it does. I stole/borrowed/learned it from German HOWTO back in 1994, when it was like the only national language oriented HOWTO. *If you have any idea about this magic sequence, please tell me*.

Finally, for those purists, who don't wont to give the *Alt* codeset a chance, I'm attaching yet another version of the script above, using native *KOI8−R* fonts.

```
if [ notset.$DISPLAY != notset. ]; then
    echo "`basename $0`:  cannot run under X"
    exit
fi

loadkeys /usr/lib/kbd/keytables/ru.map
setfont /usr/lib/kbd/consolefonts/koi-8x16
echo "Use the right Ctrl key to switch the mode..."
```

However, don't expect nice borders in your text mode−based windowing applications.

Now you probably want to test it. Do the appropriate bash or tcsh setup, rerun it, then press the right `Control` key and make sure you are getting the cyrillic characters right. The '`q`' key must produce russian "`short i`" character, '`w`' generates "`ts`", etc.

If you've screwed something up, the very best thing to do is to reset to the original (that is, US) settings. Execute the following commands:

```
loadkeys /usr/lib/kbd/keytables/defkeymap.map
setfont /usr/lib/kbd/consolefonts/default8x16
```

NOTE: unfortunately enough, the console driver is not able to preserve it's state (at least easily enough), while running the X Window System. Therefore, after you leave the X (or switch from it to a console), you have to reload the console russian font.

FreeBSD Console

I am not using FreeBSD so I couldn't test the following information. All data in this section should be treated as just pointers to begin with. The FreeBSD project homepage may have some information on the subject. Another good source is the relcom.fido.ru.unix newsgroup. Also, check the resources listed in section resources .

Anyway, this is what Ilya K. Orehov suggests to do in order to make FreeBSD console speak Russian:

1. In `/etc/sysconfig` add:

```
keymap=ru.koi8-r
keyrate=fast
# NOTE: '^[' below is a single control character
keychange="61 ^[[K"
cursor=destructive
scrnmap=koi8-r2cp866
font8x16=cp866b-8x16
font8x14=cp866-8x14
font8x8=cp866-8x8
```
2. In `/etc/csh.login`:

```
setenv ENABLE_STARTUP_LOCALE
setenv LANG ru_SU.KOI8-R
setenv LESSCHARSET latin1
```
3. Make analogous changes in `/etc/profile`

3.2 The X Window System

Like the console mode, the X environment also requires some setup. This involves setting up the input mode and the X fonts. Both are being discussed below.

The X fonts.

First of all, you have to obtain the fonts having the Cyrillic glyphs at the appropriate positions.

If you are using the most recent X (or XFree86) distribution, chances are, that you already have such fonts. In the late 1995, the X Window System incorporated a set of Cyrillic fonts, created by Cronyx. Ask your system administrator, or, if *you* are the one, check your system, namely:

1. Run '`xlsfonts | grep koi8`'. If there are fonts listed, your X server is already aware about the fonts.
2. Otherwise, run

```
find -name crox\*.pcf\*
```
to find the location of the Cyrillic fonts in the system. You'll have to `enable` those fonts to the X server, as I explain below.

If you haven't found such fonts installed, you'll have to do it yourself.

There is some ambiguity with the fonts. XFree86 docs claim that the russian fonts collection included in the distribution is developed by Cronyx. Nevertheless, you may find another set of Cronyx Cyrillic fonts on the net (eg. on ftp.kiae.su), known as the **xrus** package (don't confuse it with the `xrus` program, which is used to setup a Cyrillic keyboard layout. Hopefully, tha letter one was renamed to **xruskb** recently). **Xrus** has fewer fonts than the collection in Xfree86 (38 vs 68), but the latter one didn't go along with my Netscape setup − it gave me some really huge font in the menubar. The **xrus** package doesn't have this problem.

I would suggest you to download and try both of them. Pick up the one which you'll like more. Also, I'm going to creat RPM packages soon for both collections and download them to ftp.redhat.com.

There are also older stuff, for example the **vakufonts** package, created by Serge Vakulenko, which was the base for the one in the X distribution. There are also a number of others. The important point is that the fonts' names in the old collection were not strictly conforming to the standard. The latter is fine in general, but sometimes it may cause various weird errors. For example, I had a bad experience with Maple V for Linux, which crashed mysteriously with the **vakufonts** package, but ran smoothly with the "standard" ones.

So, let's start with the fonts:

1. Download the appropriate fonts collection. The package for XFree86 may be found at any FTP site, containing the X distribution, for example, directly from the XFree86 FTP site. The **xrus** package may be found on ftp.kiae.su
2. Now when you have the fonts, you create some directory for them. It is generally a bad idea to put new fonts to the already existing font directory. So, place them, to, say, `/usr/lib/X11/fonts/cyrillic` for a system−wide setup, or just create a private directory for personal use.
3. If the new fonts are in BDF format (`*.bdf` files), you have to compile them. For each font do:

   ```
   bdftopcf −o <font>.pcf <font>.bdf
   ```
   If your server supports compressed fonts, do it, using the *compress* program:

   ```
   compress *.pcf
   ```
   Also, if you do want to put the new fonts to an already existing font directory. you have to concatenate the old and the new files named `fonts.alias` in the case both of them exist.
4. Each font directory in the X must contain a list of fonts in it. This list is stored in the file `fonts.dir`. You don't have to create this list manually. Instead, do:

   ```
   cd <new font directory>
   mkfontdir .
   ```
5. Now you have to make this font directory known to the X server. Here, you have a number of options:

   - System−wide setup for XFree86. If you are running this version of X, then append the new directory to the list of directories in the file `XF86Config`. To find the location of this file, see output of `startx`. Also, see **XF86Config(4/5)** for details.
   - System−wide setup through `xinit`. Add the new directory to the `xinit` startup file. See **xinit(1x)** and the next option for details.
   - Personal setup. You have a special start−up file for the X − `~/.xinitrc` (or `~/.Xclients`, or `~/.xsession` for the RedHat users). Add the following commands to it:

   ```
   xset +fp <new font directory>
   xset fp rehash
   ```
   It is important to note that '`+fp`' means that the new fonts will be added to the head of the font path list. That is, if an application requests say a `fixed` font, it'll be given the one with Cyrillic characters, which is definitely what we are trying to achieve. There are problems, though. The

   `fixed` font in the cyrillic fonts distribution doesn't have it's bold and italic counterparts. My font of choice is `6x13`, so, since it also lacks bold and italic typefaces, I cannot use Emacs/XEmacs faces in their full glory. Hopefully somebody will ultimately create those fonts and the situation will change.
6. Now restart your X. If you have done everything right, the tests in the beginning of the section will be successful. Also, play with **xfontsel(1x)** to make sure you are able to select the cyrillic fonts.

In order to make the X clients use the Cyrillic fonts, you have to set up the appropriate X resources. For example, I make the russian font the default one in my `~/.Xdefaults`:

```
*font:          6x13
```

Since my cyrillic fonts are first in the font path (see output of '`xset q`'), the font above is taken from the "cyrillic" directory.

This just a simple case. If you want to set the appropriate part of the X client to a cyrillic font, you have to figure out the name of the resource (eg. using **editres(1x)**) and to specify it either in the resource database, or in the command line. Here go some examples:

```
$ xterm -font '-cronyx-*-bold-*-*-*-19-*-*-*-*-*-*-*'
```

...will run xterm with some ugly font; and

```
$ xfontsel -xrm '*quitButton.font: -*-times-*-*-*-*-13-*-*-*-*-*-koi8-*'
```

...will set a Cyrillic Times font for the **Quit** button in `xfontsel`.

The input translation

In the newest X releases (X11R61 and higher) there are two "standard" input methods: the original one, working through the **xmodmap** utility, and the new one called *Xkb* (X KeyBoard). The very first thing you have to do is **to disable the Xkb method!** Don't get charmed by it's ability to set up a "russian keyboard". It looks like this method is using the Cyrillic keysyms defined in `keysymdef.h`. This file defines keysyms for many languages. The only problem is that those definitions have nothing to do with the extended ASCII codeset − the one most programs are only able to operate with! I hardly know any programs being able to grok the `keysymdef.h` keysyms, different from 8−bit ASCII. However our goal is to get the KOI8−R support to work.

To disable the `Xkb` support, browse through the `Keyboard` section of your `XF86Config` file and comment all lines starting with *Xkb* (case doesn't matter). Instead, put the following line:

```
XkbDisable
```

The `xmodmap` program.allows customization of codes emitted by various characters and their combinations. It sets the things up based on the file containing the translation table.

In the previous versions of this document I used to describe the xmodmap−based setup in a great detail. This proved to be almost useless. The Xmodmap−based input translation method is well known as being it is non−portable, inflexible, and incomplete. Your configuration may work with one XFree version and fail with a different one. Even worse, sometimes things differ accross different servers in the same distribution.

I strongly suggest you not to play with this xmodmap, at least for now. Apart from headache and disappointment you'll gain nothing. Instead, I recommend installing the xruskb package, which allows you to configure most of the input translation parameters without having to know about xmodmap. Again, the RedHat Linux users are free to download and install an RPM package.

3.3 First steps − Cyrillic in shells

3.4 bash

Three variables should be set on order to make bash understand the 8−bit characters. The best place is ~/.inputrc file. The following should be set:

```
set meta-flag on
set convert-meta off
set output-meta on
```

3.5 csh/tcsh

The following should be set in .cshrc:

```
setenv LC_CTYPE iso_8859_5
stty pass8
```

If you don't have the POSIX stty (impossible for Linux), then replace the last call to the following:

```
stty -istrip cs8
```

3.6 ksh

As for the public domain `ksh` implementation − `pdksh 5.1.3`, you can input 8 bit characters only in `vi` input mode. Use:

```
set -o vi
```

3.7 less

So far, `less` doesn't support the KOI8−R character set, but the following environment variable will do the job:

```
LESSCHARSET=latin1
```

3.8 mc (The Midnight Commander)

To display Cyrillic text correctly, select the *full 8 bits* item in the **Options/Display** menu.

If your problem is the ugly windows' borders, consult the linux−console section.

As an off−topic, if you want to make **mc** use color in an `Xterm` window, set the variable `COLORTERM`:

```
COLORTERM= ; export COLORTERM
```

3.9 rlogin

Make sure that the shell on the destination site is properly set up. Then, if your `rlogin` doesn't work by default, use '`rlogin -8`'.

3.10 zsh

Use the same way as with `csh` (see section csh). The startup files in this case are `.zshrc` or `/etc/zshrc`.

4.Editing text

In this section I'll describe how to customize various text editors to work with Cyrillic text. This doesn't cover the *word processors*, which will be described later (see section word–processingword–processing).

4.1 Emacs and XEmacs

There are two version of the Emacs editor – **GNU Emacs** and **XEmacs**. While they provide more or less same functionality, some implementation details are significantly different. Cyrillic setup requires some low–level (in Emacs Lisp sense) tweaking, and it differs a bit for those two versions.

NOTE: Apart from the setup described here, there is an alternative way to configure both versions of emacs – use **MULE** (MULtilanguage Emacs support). The latter way is fairly complicated and (to the best of my knowledge) rarely used, so I don't discuss it here.

The minimal cyrillic support in **GNU emacs** (you don't have to do it for the **XEmacs**) is done by adding the following calls to one's `.emacs` (provided that the Cyrillic character set support is installed for console or X respectively):

```
(standard-display-european t)

(set-input-mode (car (current-input-mode))
   (nth 1 (current-input-mode))
   0)
```

This allows the user to view and input documents in Russian.

However, it isn't enough. Emacs doesn't know yet, that Cyrililic characters may constitute a word, let alon the upper/lower case conversion rules. In order to teach Emacs doing that, you have to modify the syntax and case tables of emacs:

```
(require 'case-table)

(let* ((ruc "\341\342\367\347\344\345\263\366\372\351\352\353\354\355\356\357\360\362\363\364\365\346\350\343\376\373\375\370\371\377\374\340\361")
       (rlc "\301\302\327\307\304\305\243\326\332\311\312\313\314\315\316\317\320\322\323\324\325\306\310\303\336\333\335\330\331\337\334\300\321")
       (i 0)
```

```
      (len (length ruc)))
  (while (< i len)
    (modify-syntax-entry (elt ruc i) "w  ")
    (modify-syntax-entry (elt rlc i) "w  ")
    (set-case-syntax-pair (elt ruc i) (elt rlc i) (standard-case-table))
    (setq i (+ i 1))))
```

For this purpose I created a `rusup.el` file which does this, as well as a couple handy functions. You have to load it in your `~/.emacs`.

Finally, the russian.el package by Valery Alexeev (`valery@math.uga.edu`) allows the user to switch between cyrillic and regular input mode and to translate the contents of a buffer from one Cyrillic coding standard to another (which is especially useful while reading the texts imported from MS−DOS or Windows).

4.2 Using vi

The **vi** editor (at least it's clone **vim**, available in most Linux distributions) is aware of 8−bit characters. It will allow you to enter cyrillic characters and will be able to recognize the word boundaries correctly. I don't know about the upper−/lower−case conversion rules, since I don't use **vi** much. *If you know something about it, please inform me*.

4.3 Editing text with joe

Joe requires a special `-asis` option to recognize 8−bit characters. You may either specify this option at the command line, or to put it in `~/.joerc` file (for personal use, or in `/usr/lib/joerc` for system−wide setup.

If your program doesn't understand `-asis` option, you have to upgrade to the newer version.

However, **joe** doesn't seem to understand the cyrillic words' boundaries correctly. I assume, that it applies both to the case conversion rules.

4.4 Spell−checking Russian

The program I use to spell−check text is the **GNU ispell**. It is very flexible and extensible, so it is possible to use it to spell−check text in languages, other than English, by adding new *spell dictionaries*.

Constantine Knizhnik has created a very good Russian dictionary for **ispell**. You may find it at his homepage. The distribution includes a handy incremental spelling script for **emacs**.

Ideally, if you already have an **ispell** properly installed, you have to just step into the newly−created directory and generate the dictionary, using the commands provided in the `Makefile`. However, chances are quite high, that you'll see a lot of complaints about the **ispell**'s unawareness of the 8−bit data. This is because in most distributions, **ispell** is compiled without 8−bit data support. In this case, you cannot avoid recompiling the **ispell** package.

Again, RedHat users will be delighted to know that I've rebuilt the **ispell** package with both Russian and German dictionaries. As usual, you may grab it from the RedHat FTP site.

Once you have everything installed, you may invoke Russian spell−check, by supplying `'-d russian'` option to **ispell**.

Now, if you use **Emacs**, you may want to add a menu item for a russian dictionary. I sent a proposed menu entry to the `ispell.el` maintainer and he kindly agreed to include it in the the next public release of the file. Meanwhile, you may do it by adding the following code in your `~/.emacs` (or in `/usr/share/emacs/site-lisp/site-start.el` for a system−wide setup):

```
(setq ispell-dictionary-alist
  (append ispell-dictionary-alist
          '(("russian"
             "[\341\342\367\347\344\345\263\366\372\351\352\353\354\355\356\357\360\362\363\364\365\346\350\343\376\373\375\370\371\377\374\340\361\301\302\327\307\304\305\243\326\332\311\312\313\314\315\316\317\320\322\323\324\325\306\310\303\336\333\335\330\331\337\334\300\321]"
             "[^\341\342\367\347\344\345\263\366\372\351\352\353\354\355\356\357\360\362\363\364\365\346\350\343\376\373\375\370\371\377\374\340\361\301\302\327\307\304\305\243\326\332\311\312\313\314\315\316\317\320\322\323\324\325\306\310\303\336\333\335\330\331\337\334\300\321]"
             "[']" t ("-C" "-d" "russian") "~latin1"))))

(define-key-after ispell-menu-map [ispell-select-russian]
  '("Select Russian (KOI-8)" . (lambda ()
                                 (interactive)
                                 (ispell-change-dictionary "russian")))
  'british)
```

Unfortunately, it won't work for the **XEmacs**. I'll try to solve this problem later.

5.Using Cyrillic with mail and news

Setting up your mail and news software to recognize Cyrillic text is not very difficult, although you have to possess some knowledge of principles, mail and news work by.

Internet electronic mail software generally consists of two parts: **MUA** (Mail User Agent) and **MTA** (Mail Transfer Agent). MUA is the program you use to read, compose, and send mail. However, MUA doesn't transfer mail messages by itself. Instead, it calls the MTA, which is reponsible to send message using an appropriate protocol to the appropriate direction. For example, your MUA may be **Pine** and MTA − **qmail**.

Until quite recently, both MTA and MUA weren't 8−bit clean by default. Therefore, whenever you sent your message from say America to Russia, you were never sure, that some intermediate MTA won't strip the 8th bit from each character of your message. Therefore, a set of protocols was developed, which allowed encoding various kinds of data using only printable characters from 7−bit ASCII. This family of protocols is called **MIME** (MultimedIa Mail Encoding).

Since MIME is usually pre−configured to reasonable defaults, we won't describe it here. We will talk more about MIME when we provide a backward compatibility with other Cyrillic encodings (section mime).

Meanwhile, we start MUA setup, because it is usually up to an end−user. Then, we will describe the basic priciples of the MTA configuration for Cyrillic.

5.1 Setting up Mail User Agents

Emacs−based mail readers

Basically, you don't need any special setup for Emacs−based readers, geivedn, that you've already configured the emacs itself (see section emacs).

pine

Set the following directive in `~/.pinerc` for personal configuration, or in `/usr/lib/pine.conf` for a global one:

```
character-set=ISO-8859-5
```

5.2 Configuring your MTA

There are a number of MTAs available now. These include **sendmail**, **qmail**, **smail**, **exim**, and others.

sendmail

So far, **sendmail** is much more popular than other MTAs, because it's long history and widespread use. Personally, I hate this program − it is a perfect example of a completely moronic design and even it's "improvements" with the passion of time show, that this approach is not going to cease. Any system administrator shudders, when he hears the ominous "`sendmail.cf`" name...

As of now, **sendmail** doesn't strip the 8th bit anymore. However, it may *encode* the 8−bit data using a special *base64* encoding. Although most MUAs are supposed to recognize it and decode it back to a regular data, you may want to start with sending raw 8−bit text to make sure everything works.

As of version 8, **sendmail** handles 8−bit data correctly by default. If it doesn't do it for you, check the `EightBitMode` option and option `7` given to mailers in your `/etc/sendmail.cf`. See *"Sendmail. Operation and Installation Guide"* for details.

Other MTAs

I don't know much about other MTAs. If you know something, which may be important for Cyrillic setup, please inform me.

6.Browsing the Cyrillic Web

Unlike e−mail and news, there is no definitive standard for Cyrillic encoding for the Web. This is primarily because Microsoft offers Web authoring tools, which only allow *cp1251* codeset for Cyrillic, completely ignoring the fact that any other standards may already exist.

The setup described here is very basic. It will allow you to view pages in the *KOI8−R* codeset. If the situation improves, I'll add more information.

6.1 lynx

As of version 2.6, you may select the appropriate encoding for the `display Character set` option.

6.2 Netscape navigator

Make sure you are using `Netscape` version higher than 3. If your `Netscape` is older, download a new one from www.netscape.com.

Basic setup

To be able to see Cyrillic text in most parts of the HTML document, do the following:

- In menu **Options/Document Encoding** select **Cyrillic(KOI−8)**.
- In menu **Options/General Preferences/Fonts** select **Cyrillic (KOI−8)** encoding, **Times(Cronyx)** as a proportional font and **Courier(Cronyx)** as a fixed one.
- save options.

NOTE: This setup will work with most parts of the document. However, you won't be able to display Cyrillic text in the window header, menus and some controls. Attempts to fix it follows.

Cyrillic text in frames and input areas

To fix this, it is usually enough to:

1. Copy the Netscape properties database (usually `Netscape.ad`) to `~/Netscape`.
2. In the latter file, set the following property:

```
*documentFonts.charset*iso8859-1:           koi8-r
```

This will force all frame and input elements to use the fonts with *koi8−r* encoding instead of the default ones, therefore you have to make sure you have installed such fonts (see section xfonts).

The bad news about the trick above is that if you load a document which is supposed to be displayed in `iso-8859-1` fonts, it will be displayed using the `koi8` fonts instead. Sometimes such documents will look worse.

Advanced setup

Andrew A. Chernov is the one, who knows more than others about KOI−8 in general and netscape in particular. Visit his excellent KOI−8 page and download a patch for Netscape resource file, making Netscape speak Russian as much as it is able to.

7.Cyrillic wordprocessing

7.1 TeX−based environments

In this section I'll describe several ways to make TeX and LaTeX typeset Cyrillic texts. There are several ways, which differ in setup sophistication and usage convenience. For example, one possibility is to start without any preliminary setup and use the *Washington AMSTeX Cyrillic fonts*. On the other hand, you may install a LaTeX package, providing a very high degree of Cyrillic setup. I have an experience with two such packages. One is the `cmcyralt` package by Vadim V. Zhytnikov (`vvzhy@phy.ncu.edu.tw`) and Alexander Harin (`harin@lourie.und.ac.za`), and the other one is the `LH` package by the *CyrTUG* group with styles and hyphenation for LaTeX2e by Sergei O. Naoumov (`serge@astro.unc.edu`). I'll describe both.

Note, that there are two versions of LaTeX available − 2.09 is the old one, while 2e is a new pre−3.0 release. If you are using LaTeX 2.09, then switch quickly to the 2e. The latter retains compatibility with the old one, but has much more features. Hopefully, version 3 will be released soon. I describe a LaTeX 2e setup.

Also, both of these packages require the Cyrillic text to be typeset using the *Alt* codeset, not *KOI8−R*! This is caused by historical reasons, since the creators of these packages used to work with `EmTeX` − the MS−DOG version of TeX (they didn't know about Linux yet :−). Switching to the *KOI8−R* requires some effort and is being expected to be done soon. So far, use some utility to convert your russian text from *KOI8−R* to *Alt*. See section user−toolsuser−tools .

Using the Washington Cyrillic

This package was created for the American Mathematic Society to provide documents with Russian references. Therefore, the authors were not very careful and the fonts look quite clumsy. This package is usually referred to as a `"really bad cyrillic package for TeX"`.

Nevertheless, we'll discuss it, because it is very easy to use and doesn't require any setup − this collection is supplied with most of TeX distributions.

Of course, you won't be able to use such luxury as automatic hyphenation, but anyway...

1. Prepend your document with the following directives:

```
\input cyracc.def
\font\tencyr=wncyr10
\def\cyr{\tencyr\cyracc}
```

2. Now to type a cyrillic letter, you enter

```
\cyr
```

and use a corresponding latin letter or a TeX command. Thus, the lower case of the Russian alphabet is expressed by the following codes:

```
a b v g d e \"e zh z i {\u i} k l m n o p r s t u f kh c ch sh shch
{\cprime} y {\cdprime} \`e yu ya
```

It is extremely inconvenient to convert your Russian texts to such encoding, but you can automate the process. The translit program (section user−toolsuser−tools) supports a TeX output option.

KOI−8 package for teTeX

There is some new teTeX−rus package. It is reported to support KOI−8 character set and have all basic stuff required for TeX and LaTeX. I personally haven't tried it yes, although I heard about it's successfull usage.

NOTE: This package requires you to reconfigure and rebuild some parts of your **teTeX** package (for example the precompiled LaTeX macros). **Unless you know what you are doing, you shouldn't try it without necessary care. Otherwise, you may be better off by borrowing the precompiled parts fron somebody on the net**

Using the cmcyralt package for LaTeX

The `cmcyralt` package can be found on any CTAN (Comprehensive TeX Archive Network) site like `ftp.dante.de`. You should obtain two pieces: the fonts collection from `fonts/cmcyralt` and the styles and hyphenation rules from `macros/latex/contrib/others/cmcyralt`.

Note: Make sure you have the `Sauter` package installed, since `cmcyralt` requires some fonts from it. You can get this package from CTAN site as well.

Now you should do the following:

1. Put the new fonts to the TeX fonts tree. On my system (Slackware 2.2) I created a `cmcyralt` directory in the `/usr/lib/texmf/fonts/cm/`. Create the `src`, `tfm`, and `vf` subdirectories in it. Put there `.mf`, `.tfm`, and `vf` files respectively.
2. Put the font driver files (`*.fd`) from the styles archive to the appropriate place (in my case it was `/usr/lib/texmf/tex/latex/fd`).
3. Put the style files (`*.sty`) to the appropriate LaTeX styles directory (in my case `/usr/lib/texmf/tex/latex/sty`).

Now the hyphenation setup. This requires to remake the LaTeX base file.

1. The file `hyphen.cfg` contains the directives for both English and Russian hyphenation. Extract the one for Russian and place it to the LaTeX hyphenation config file `lthyphen.ltx`. In my case, that file was in `/usr/lib/texmf/tex/latex/latex-base`.
2. Put the `rhyphen.tex` to the same directory. It is needed for making the new base file. Later, you can remove it.
3. Do '`make`' in that directory. Don't for get to make a link from `Makefile` to `Makefile.unx`. During the make process check the output. There should be a message:

   ```
   Loading hyphenation patterns for Russian.
   ```
   If everything goes OK, you will get the new `latex.fmt` in that directory. Put it to the appropriate place, where the previous one was (like `/usr/lib/texmf/ini/`). **Don't forget to save the previous one!**.

This is it. The installation is complete. Try processing the examples found in the styles archive. If you are to create the PostScript files without any problems, then everything is OK. Now, to use Cyrillic in LaTeX, prepend your document with the following directive:

```
\usepackage{cmcyralt}
```

For more details, see the `README` file in the `cmcyralt` styles archive.

Note: if you do have problems with the examples, provided you have installed the things right, then probably your TeX system hasn't been installed correctly. For example, during my first try, every attempt to create the `.pk` files for the russian fonts failed (`MakeTeXPK` stage). A substantial investigation discovered some implicit conflict between the *localfont* and *ljfour*`METAFONT` configurations. It used to work before, but kept crashing after the `cmcyralt` installation. Contact your local TeX guru – TeX is very (sometimes too much) complicated to reconfigure it without any prior knowledge.

Using the CyrTUG package

You can obtain the CyrTUG package from the SunSite archive. Get the files `CyrTUGfonts.tar.gz`, `CyrTUGmacro.tar.gz`, and `hyphen.tar.Z`.

The process of installation doesn't differ from too much the previous one.

7.2 The StarOffice suite

Youri Kovalenko (http://www.inp.nsk.su/~kovalenko) has compiled a concise summary on StarOffice russification. It is located at ftp://sky.inp.nsk.su/archives_src/linux/StarOffice/russification.txt. I never had a chance to try it, so I cannot say anything about it's correctness.

Another source of information on the subject is compiled by Eugene Demidov (mailto:jack@gpi.ru) and is located at ftp://ftp.kapella.gpi.ru/pub/cyrillic/psfonts/README.

8.Printing and PostScript

8.1 Text to PostScript conversion

Sometimes you have just a plain ASCII KOI8−R text and you want to print it just to get it on the paper. One of the easiest ways to achieve that is to use special programs converting text to PostScript.

There are a number of programs doing such conversion. I personally prefer a2ps. Originally developed as a simple text−to−PostScript converter it became a big and highly configurable program with many options and allows you to manage various page layouts, syntax highlighting etc. Another tool (now available as a part of the *GNU* project) is enscript.

An a2ps converter

A text to PostScript converter has been around for a while and is one of the most versatile printing tools. The author proved to be very open to suggestions, so since the release 4.9.8 **a2ps** supports Cyrillic right off−the−shelf. All you need is a PostScript printer.

The command I use is:

```
a2ps -X koi8r --print-anyway  <file>
```

The GNU enscript

The GNU **enscript** program is also designed for converting text to PostScript and it also has a non−ASCII codeset support. It doesn't have Cyrillic PostScript fonts, but it is very easy to get them, as will be explained below (thanks to Michael Van Canneyt):

1. Install the newest **enscript**. As of now, the most recent release is 1.5. You may either get the one from the GNU FTP archive, or take an RPM package from the Redhat site.

2. Now, if you are a lucky RedHat Linux user, download and install Cyrillic Textbook font.
3. If you don't use RPM, download a file `textbook.tar.gz` from the Cyrillic Software collection on sunsite.unc.edu. Extract it to a directory, where **enscript** fonts are located (usually `/usr/share/enscript`). Now change to that directory and run the following command:

```
mkafmmap *.afm
```
4. The setup is finished. Try to print some text in KOI8−R Cyrillic with the following command:

```
enscript --font=Textbook8 --encoding=koi8 some.file
```

If you want a really quick and dirty solution and you don't care about the output quality and all you need is just Cyrillic on the paper, try the rtxt2ps package. It is a very simple no−frills text−to−PostScript conversion program. The output quality is not very good (or, to be honest, just *bad*) but it does it's job.

8.2 Text to TeX conversion

If all you need is just to print an ASCII text without any additional word processing, you may try to use some programs, which would convert your Cyrillic text to a ready−to−process TeX file. One of the best programs for such purposes is **translit** (see section conversion). In this case, you don't even have to bother about installing the Cyrillic fonts for TeX, since **translit** uses a *Washington Cyrillic* package, which is included in most TeX distributions (or am I wrong?)

9.Cyrillic in PostScript

Experts say PostScript is easy. I cannot judge − I've got too many things to learn to spare some time to learn PostScript. So I'll try to use my sad experience with it. **I'll appreciate any feedback from you guys who know more on the subject than I do** (approx. 99% of the Earth population).

Basically, in order to print a Cyrillic text using PostScript, you have to make sure about the following things:

- Cyrillic font is *loaded* or included in the document.
- Cyrillic text is included in the document.
- Cyrillic text uses the appropriate character codes which correspond to the font's requirements.
- An appropriate font is *selected* in order to print Cyrillic text.

There is no solution general enough to be recommended as an ultimate treatment. I'll try to outline various ways to cope with different problems related to the subject.

One way to address Cyrillic setup problems generally enough is to use Ghostscript. **Ghostscript** (or just **gs** in the newspeak) is a free (well quasi−free) PostScript interpreter. It has many advantages; among them:

- Ability to run on many platforms (various Unices, Windows etc)
- Support for a wide number of non−PostScript printers
- Good degree of configurability

What is important in our particular case, is that once **Ghostscript** is set up, we can do all printing through it, thus eliminating extra setup for other PostScript devices (for example *HP LaserJet IV*)

9.1 Adding Cyrillic fonts to Ghostscript

This is important, since you probably don't want to put a responsibility to other programs to insert Cyrillic fonts in the PostScript output. Instead, you add them to **gs** and just make the programs generate Cyrillic output compatible with the fonts.

To add a new font (in `pfa` or `pfb` form) in **gs**, you have to:

1. Put it in the **gs** fonts directory (ie. `/usr/lib/ghostscript/fonts`).
2. Add the appropriate names and aliases for the font in the `Fontmap` file in the **gs** directory.

Recently a decent set of Cyrillic fonts for **GhostScript** appeared. It is located in ftp.kapella.gpi.ru. This one even has a necessary part to add to the `Fontmap` file. You have to download the contents of the `/pub/cyrillic/psfonts` directory. The `README` file describes the necessary details.

10.Print setup

Printing is always tricky. There are different printers from different vendors with different facilities. Even for a native printing there is no uniform solution (this applies not only to UNIX, but to other operating systems as well.

Printers have different control languages and often they have very different views on foreign language support. The good news is that on control language seems to be recognized as a de−facto standard for print job description − it is a PostScript language developed by Adobe Corporation.

Another problem is a variety of requirements to the print services. For example, sometimes you want just to print a piece if C program, containing comments in Russian, so you don't need any pretty−printing − just a raw ASCII output in a single font. Another time, when you design a postcard for your girlfriend, you'll probably need to typeset some document with different fonts etc. This will definitely require more effort to setup Cyrillic support.

To accomplish the former task you just have to make your printer understand *one* Cyrillic font and (maybe) install some filter program to generate data in appropriate format. To accomplish the latter one, you have to teach your printer different fonts and have a special software.

There is also something in the middle, when you get a program which knows how to generate both the fonts and the appropriate printer input, so you can say do some aource code pretty−printing without sophisticated word processing systems.

All these options will be more or less covered below.

10.1 Pre−loading Cyrillic fonts into a non−PostScript printer

If you have a good old dot matrix printer and all you need is to print a raw KOI8−R text, try the following:

1. Find a proper KOI8−R font for your printer. Check out the MS−DOSish stuff on the Internet (for example the SimTel archive).
2. Learn from the manual, how to load such font into your printer and, probably, write a simple program doing that.
3. Run this program from the appropriate `rc` file at a boot time.

Thus, having Cyrillic characters in the upper part of the printer's character set will allow you to print you texts in Russian without any hussle.

Alternatively to the *KOI8−R* fonts you may try to use the *Alt* font. There are two reasons for that:

- It may be probably much easier to find an *Alt* font, since those were very widespread in the MS−DOS culture.
- Having a proper *Alt* font will allow you to print pseudo−graphic characters as well.

However in this case, you'll have to convert your texts from *KOI8−R* to *Alt* before sending them to a printer. This is quite easy, since there are a lot of programs doing that (see user−toolstranslit for example), so you just have to call such program properly in the `if` field in `/etc/printcap` file. For example, with the **translit** program you may specify:

```
if=/usr/bin/translit −t koi8−alt.rus
```

See **printcap(5)** for details.

10.2 Printing with different fonts

One great way to cope with different printers and fonts is to use **TeX** (see section tex). TeX drivers handle all details, so once you make TeX understand Cyrillic fonts, you are done.

Another possibility is to use *PostScript*. I decided to devote an entire chapter postscript to the subject, since it is not simple.

Finally, there are other word processors, which have printer drivers. I never tried anything apart from TeX, so I cannot suggest anything.

11.Localization and Internationalization

So far, I described how to make various programs understand Cyrillic text. Basically, each program required it's own method, very different from the others. Moreover, some programs had incomplete support of languages other than English. Not to mention their inability to interact using user's mother tongue instead of English.

The problems outlined above are very pressing, since software is rarely developed for home market only. Therefore, rewriting substantial parts of software each time the new international market is approached is very ineffective; and making each program implement it's own proprietary solution for handling different languages is not a great idea in a long term either.

Therefore, a need for standardization arises. And the standard shows up.

Everything related to the problems above is divided by two basic concepts: *localization* and *internationalization*. By localization we mean making programs able to handle different language conventions for different countries. Let me give an example. The way date is printed in the United States is MM/DD/YY. In Russia however, the most popular format is DD.MM.YY. Another issues include time representation, printing numbers and currency representation format. Apart from it, one of the most important aspect of localization is defining the appropriate character classes, that is, defining which characters in the character set are language units (letters) and how they are ordered. On the other hand, localization doesn't deal with fonts.

Internationalization (or *i18n* for brevity) is supposed to solve the problems related to the ability of the program interact with the user in his native language.

Both of the concepts above had to be implemented in a standard, giving programmers a consistent way of making the programs aware of national environments.

Althogh the standard hasn't been finished yet, many parts actually have; so they can be used without much of a problem.

I am going to outline the general scheme of making the programs use the features above in a standard way.

Since this deserves a separate document, I'll just try to give a very basic description and pointers to more thorough sources.

11.1 Locale

One of the main concept of the localization is a *locale*. By locale is meant a set of conventions specific to a certain language in a certain country. It is usually wrong to say that locale is just country−specific. For example, in Canada two locales can be defined − Canada/English language and Canada/French language. Moreover, Canada/English is not equivalent to UK/English or US/English, just as Canada/French is not equivalent to France/French or Switzerland/French.

How to use locale

Each locale is a special database, defining at least the following rules:

1. character classification and conversion
2. monetary values representation
3. number representation (ie. the decimal character)
4. date/time formatting

In RedHat 4.1, which I am using there are actually *two* locale databases: one for the C library (`libc`) and one for the `X` libraries. In the ideal case there should be only one locale database for everything.

To change your default locale, it is usually enough to set the `LANG` environment variable. For example, in **sh**:

```
LANG=ru_RU
export LANG
```

Sometimes, you may want to change only one aspect of the locale without affecting the others. For example, you may decide (God knows why) to stick with `ru_RU` locale, but print numbers according to the standard POSIX one. For such cases, there is a set of environment variables, which you can you to configure specific parts for the current locale. In the last exaple it would be:

```
LANG=ru_RU
LC_NUMERIC=POSIX
export LANG LC_NUMERIC
```

For the full description of those variables, see **locale(7)**.

Now let's be more Linux−specific. Unfortunately, Linux `libc` version 5.3.12, supplied with RedHat 4.1, doesn't have a russian locale. In this case one must be downloaded from the Internet (I don't know the exact address, however).

To check, locale for which languages you have, run '`locale -a`'. It will list all locale databases, available to libc.

Fortunately, Linux community is rapidly moving to the new GNU libc (`glibc` version 2, which is much more POSIX−compliant and has a proper russian locale. Next "stable" RedHat system will already use `glibc`.

As for the `X` libraries, they have their own locale database. In the version I am using (`XFree86 3.3`), there already is a russian locale database. I am not sure about the previous versions. In any case, you may check it by looking into usr/lib/X11/locale/ (on most systems). In my case, there already are subdirectories named `koi8-r` and even `iso8859-5`.

Locale−aware programming

With locale, program don't have to implement explicitly various character conversion and comparison rules, described above. Instead, they use special API which make use of the rules defined by locale. Also, it is not necessary for program to use the same locale for all rules − it is possible to handle different rules using different locales (although such technique should be strongly discouraged).

From the **setlocale(3)** manual page:

> A program may be made portable to all locales by calling `setlocale(LC_ALL, "" )` after program initialization, by using the values returned from a `localeconv()` call for locale − dependent information and by using `strcoll()` or `strxfrm()` to compare strings.

SunSoft, for example, defines 5 levels of program localization:

1. *8−bit clean* software. That is, the program calls `setlocale()`, it doesn't make any assumptions about the 8th bit of each character, it users functions from `ctype.h` and limits from `limits.h`, and it takes care about `signed/unsigned` issues. It is very important *not* to do any assumption about the character set nature and ordering. The following programming practices must be avoided:

   ```
   if (c >= 'A' && c <= 'Z') {
         ...
   ```

   Instead, macros from the `ctype.h` header file are locale−aware and should be used in all such occasions.
2. Formats, sorting methods, paper sizes. The program uses `strcoll()` and `strxfrm()` instead of `strcmp()` for strings, it uses `time()`, `localtime()`, and strftime()/ for time services, and finally, it uses `localeconv()` for a proper numbers and currency representation.
3. Visible text in message catalogs. The program must isolate all visible text in special *message catalogs*. Those map strings in English to their translation to other languages. Selection of messages

in an appropriate for a particular environment language is done in a way which is completely transparent for both the program and it's user. To make use of those facilities, the program must call `gettext()` (Sun/POSIX standard), or `catgets()` (X/Open standard). For more information on that see section i18n .
4. EUC/Unicode support. At this level, the program doesn't use the `char` type. Instead it uses `wchar_t`, which defines entities big enough to contain Unicode characters. ANSI C defines this data type and an appropriate API.

For a more detaled explanation of locale, see, for example (Voropay1) or (SingleUnix).

11.2 Internationalization

While localization describes, how to adapt a program to a foreign environment, *internationalization* (or *i18n* for brevity) details the ways to make program communicate with a non−English speaking user.

Before, that was done by developing some abstraction of the messages to output from the program's code. Now, such mechanism is (more or less) standardized. And, of course, there are free implementations of it!

The GNU project has finally adopted the way of making the internationalized applications. Ulrich Drepper (`drepper@ipd.info.uni-karlsruhe.de`) developed a package `gettext`. This package is available at all GNU sites like prep.ai.mit.edu. It allows you to develop programs in the way that you can easily make them support more languages. I don't intend to describe the programming techniques, especially because the `gettext` package is delivered with excellent manual.

Request for collaboration: If you want to learn the `gettext` package and to contribute to the GNU project simultaneously; or even if you just want to contribute, then you can do it! GNU goes international, so all the utilities are being made locale−aware. The problem is to translate the messages from English to Russian (and other languages if you'd like). Basically, what one has to do is to get the special `.po` file consisting of the English messages for a certain utility and to append each message with it's equivalent in Russian. Ultimately, this will make the system speak Russian if the user wants it to! For more details and further directions contact Ulrich Drepper (drepper@ipd.info.uni−karlsruhe.de).

12.Staying compatible

Being standard is not the only issue. To be really nice, one has to provide the backward compatibility. In our case, this means that the configuration should be tolerant to the data created using non−standard character sets − that is the *Alt (cp866)* and *cp1251* ones. Also, we should be able to run Cyrillic programs for MS−DOS.

In most cases (except for HTTP), it is enough to provide a timely conversion of data to *KOI8−R*. When we talk about raw unstructured data, it is quite trivial − see section user−toolsConversion Utilities.

Another issue is the structured data. This case is more tricky. I'll try to outline the basic roadmap of fixing it.

12.1 MIME−based data compatibility

MIME is a standard for architecture−independent data representation. Originally developed for mail messages, it has now many more applications. MIME defines format, which is open to extensions and allows architecture−specific handling of data. For example, if I receive a mail message, containing a *MIME object* of the `video/mpeg` type (an encoded MPEG file), my mail reader will automatically decode it and start an MPEG player.

Most UNIX programs, offering MIME capabilities, are based on the **metamail** package, which contains a set of utilities and data files to work with MIME objects. Several configuration files (`/etc/mailcap` for global usage and `~/.mailcap` for personal setup) define rules for handling MIME object of various types.

Thus, if you receive a proper MIME data stream, containing text in one of the obsolete character sets, you may define a MIME rule to convert such text to KOI8.

Below a number of MIME rules are shown, which are supposed to handle plain text and richtext objects, using both of the obsolete codesets, discussed above. You may incorporate these rules into one of the MIME configuration files.

Note, that these rules use the **translit** package to perform the actual conversion. For more information on that program and the conversion in general see section user−toolsConversion Utilities.

```
text/plain; translit -t cp1251-koi8.rus < %s; test=test \
    "`echo %{charset} | tr '[A-Z]' '[a-z]'`"  = cp1251; copiousoutput

text/richtext; translit -t cp1251-koi8.rus < %s; test=test \
    "`echo %{charset} | tr '[A-Z]' '[a-z]'`"  = cp1251; copiousoutput

text/plain; translit -t alt-koi8.rus < %s; test=test \
    "`echo %{charset} | tr '[A-Z]' '[a-z]'`"  = cp866; copiousoutput

text/richtext; translit -t alt-koi8.rus < %s; test=test \
    "`echo %{charset} | tr '[A-Z]' '[a-z]'`"  = cp866; copiousoutput

text/plain; translit -t alt-koi8.rus < %s; test=test \
    "`echo %{charset} | tr '[A-Z]' '[a-z]'`"  = alt; copiousoutput

text/richtext; translit -t alt-koi8.rus < %s; test=test \
    "`echo %{charset} | tr '[A-Z]' '[a-z]'`"  = alt; copiousoutput
```

Obviously enough, this will work for plain text data only. Binary files are supposed to handle the codeset issues themselves (at least their "parent" applications are). Therefore, if you receive a Microsoft Word document in the *cp1251* character set, the duty of providing appropriate conversion capabilities lays upon an application you use to read that document (for example Microsoft Word, or Applix Words).

Unfortunately, the real situation is not that ideal. Many application have their own idea on how to use MIME. Until recently Microsoft Mail software had a broken MIME engine. Also, the Netscape Navigator/Communicator mail client is notorious because of it's sending of mail messages, encoded in *cp1251* with the *charset=koi8−r* field in the message header and vice versa.

12.2 Explicit character set conversion

There are a lot of conversion routines for Cyrillic on the Internet. Each of them has it's own quirks and it's own degree of Cyrillic support.

In my opinion tools must be standard. In this particular case the "standard" conversion tool is **GNU recode**. Unfortunately, the version, found on the official GNU site (3.4) doesn't support Cyrillic yet (only *ISO−8859−5*). I developed a set of conversion tables for *KOI8−R*, *Alt*, and *cp1251* for **recode** and submitted them to the **recode** maintainer. He promised to provide Cyrillic support in the upcoming release. Once it happens, I'll rewrite this section to recommend **GNU recode** as the standard conversion engine for Cyrillic.

Meanwhile, I would recommend a translit package. It supports many popular codesets and is even able to produce a *TeX files (see section tex) from text in Russian. Also, RedHat users will enjoy an RPM package for translit.

For other conversion routines, Look at SovInformBureau or ftp.funet.fi. You can even use the special mode for `emacs` (see section Emacs).

12.3 Cyrillic in the DOS emulator

This seems to be the only application, which may require `Alt` Cyrillic character set. The reason is that `Alt` is native to DOS and most of DOS programs dealing with Cyrillic are `Alt`−oriented.

For the console version (`dos`) you just have to load a keyboard and screen driver. Most of DOS drivers will work fine. I personally use the **rk** driver by A. Strakhov, which works for both console and X versions of **dosemu**. Another choice is the `r` driver by V. Kurland (sorry for possible misspelling). It is perfectly customizable and supports many codesets, `Alt` and `KOI8` among them. However it won't work for the X window (at least version 1.14 I'm using).

Both drivers can be found on most Russian Internet sites, for example Kurchatov Institute FTP server.

For the X version of **dosemu** you have to provide an appropriate X font as well. Alex Bogdanov sent me such font by e−mail. It is an original `vga` font from the **dosemu** distribution, modified for the `Alt` codeset. Unfortunately I don't know who is the creator of this font and where the official site is.

To setup the font for `dosemu` you should

- Introduce this font to the X. This is described in X fonts setup.
- Introduce this font to `dosemu`. If the font just replaces the original `vga` font, then it will be recognized by default. Otherwise, you have to describe it in `/etc/dosemu.conf`:

```
# Font to use (without filename extensions). For example:
X { updatefreq 8 title "MS DOS" icon_name "xdos" font "vga-alt"}
```

Finally, you have to load a keyboard driver. Note, the you don't need a screen driver for the X window. Therefore, not all drivers will work. At least two will: `rk` by A. Strakhov, and `cyrkeyb` by Pete Kvitek.

13.Bibliography

1. Andrey Chernov. KOI−8. KOI−8 information and setup.
2. Ulrich Drepper. Internationalization in the GNU project. Very thorough description of a GNU approach to i18n.
3. Michael Karl Gschwind. Internationalization. Various resources on i18n.
4. Sergei Naumov. Information on Cyrillic Software. Cyrillic setup information.
5. The Open Group Single UNIX specification.
6. RFC 1489 RFC 1489
7. Alec Voropay. Localization as it is. General locale usage in Russian.

14.Summary of the various useful resources

a2ps homepage

General Linux Information

Collection of Cyrillic resources

Cyrillic resources at KIAE

Cyrillic resources at RELCOM

Cyrillic resources at FUNET

Cronyx − the creators of Cyrillic fonts for the X Window System.

Cyrillic fonts for Ghostscript and StarOffice

Cyrillic fonts for X

Ghostscript

GNU enscript

relcom.fido.ru.linux newsgoup.

relcom.fido.ru.unix newsgoup.

Russian dictionary for GNU ispell

SovInformBureau

teTeX russification package

The kbd package for Linux

The remap package for Emacs

The rtxt2ps package

The russian.el package for emacs

The translit package

The xruskb package

Useful Cyrillic packages

X fonts collections

XFree86 FTP site