Can R Speak Your Language?

Brian D. Ripley

Professor of Applied Statistics
University of Oxford

ripley@stats.ox.ac.uk
http://www.stats.ox.ac.uk/∼ripley

Languages

The *lingua franca* of computing is (American) English.

R uses English for its keywords and its (several thousand) built-in functions.

However, the *data* used in computing can be in any human language (or none). One issue is to *represent* that data: we will discuss *character sets* (aka *charsets*) shortly. Another is to display the data in a human-readable form, both on the console and on graphics devices.

Part of the data is information related to the packages themselves. This includes the manuals, help files and so on. The terser the information, the harder it is for non-native speakers to comprehend: this applies especially to menus and information/warning/error messages.

Internationalization (aka *i18n*) is the process of enabling support for data and messages in different languages.
Localization (aka *i10n*) is the process of adding language-specific features, such as translations of messages.

Character Sets

Long ago encodings such as EBCDIC and ASCII were developed to represent characters in decimal and binary computers. The 'A' abbreviates American, and these only handled upper- and lower-case letters A-Z, digits, punctuation symbols, American currency ($ but not cent), and some 'odds' such as `# % * + / \ ^ _ | ~`.

Characters *vs* glyphs

A character is an abstract concept, and a glyph is a visual representation of it.

Did ASCII mean minus or hyphen by -? It matters both in interpretation and display in proportionally-spaced fonts.

What is '? An apostrophe, a quotation mark, an accent?

What is ‘? A quotation mark or an accent?

What is ~? A tilde accent or more like ∼?

Character Sets II

People writing non-American English (even UK English) need other characters sets. Initially there were many proprietary extensions (DEC, HP, MacRoman, Windows), but a fairly small number of character sets became common. These represent up to 255 characters, using a single 8-bit byte for each character. Examples include ISO 8859-1 (aka Latin 1) and KOI8-R.

Single-byte encodings cover most of the world's languages. (Vietnamese uses perhaps the most characters of these.) The exceptions are Chinese, Japanese, Korean (CJK) and languages with diacritical marks like Thai. CJK languages have tens of thousands of characters (and more glyphs). A number of 'shift' coding systems were developed, such as EUC-JP and the Windows code pages CP932, 936, 949 and 950.

Note that these character sets only allow a small set of languages at a time (in addition to English), not for example German and Polish.

Unicode

The grand vision was to have a character set to represent all human languages at once that would be used everywhere.

The nearest we have in *Unicode*. This has 2^{21} slots for characters, and most have now been filled with human languages and other forms of notation.

That leaves the question of how those 2^{21} slots are represented as data. Two main ways have emerged.

UTF-8 Each character is represented by one to four bytes, with ASCII using one. [Used by most Unix-alikes, but only recently.]

UCS-2 Most characters are represented by two bytes, those with code over 2^{16} by four. [Used by NT-based Windows for a decade, but characters beyond 2^{16} are turned off by default.]

Unfortunately, OSes that support one tend not to support the other, so R has to cope with both, as well as with older OSes (e.g. Windows 98) that support neither.

Internationalization of R

The major task was to add support of multi-byte character sets (MBCS), which was added in R 2.1.0 in April 2005. By that time Linux systems set up to use UTF-8 were becoming common, and MacOS X is in part based on UTF-8. The task was non-trivial, as all operations on character strings had to be reviewed and if necessary converted to work with characters and not bytes.

At the same time, the standard GNU `gettext` facilities were grafted onto R to provide for the translation of messages. Again, this was a larger task that might be supposed as the `gettext` system was not designed for an extensible system, and also for a Unix-alike rather than Windows. Least trivial of all, someone had to mark up messages for translation, and the opportunity was taken to rationalize and improve the error messages.

Also at the same time, `iconv` was added to allow translation between character sets (as seamlessly as possible) as well as mark-up for the character encodings used in documentation.

Localization of R

Once translation facilities were in place, volunteers set up translation teams. We now have partial translations for

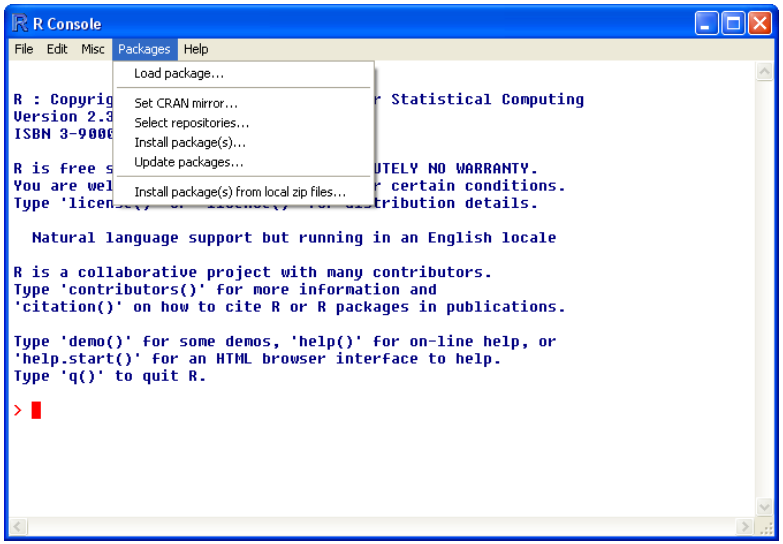
> Brazilian Portuguese, Chinese (both Simplified and Traditional orthography), English (as opposed to American), French, German, Italian, Korean, Russian and Spanish.

and it is an on-going effort to keep these current.

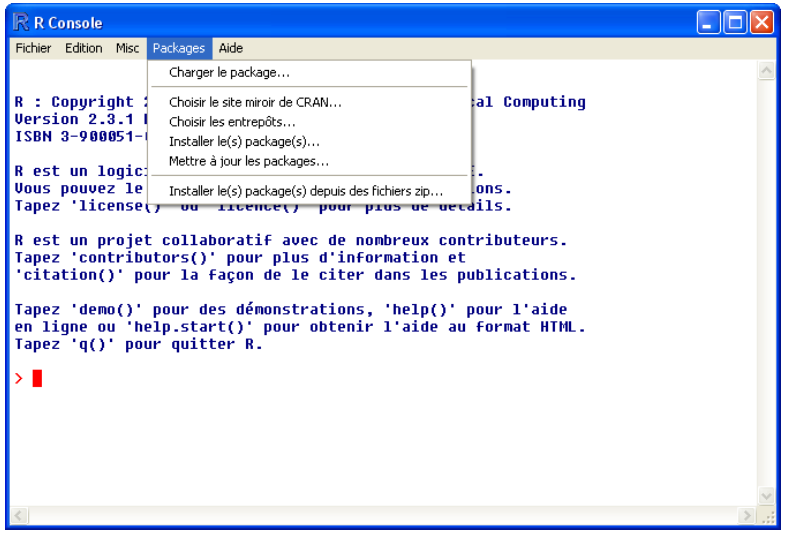
There are separate localization efforts for the Windows installer and for the MacOS X console (Dutch, French, German, Italian and Japanese).

Another issue for the CJK languages is that they are displayed with wider glyphs than ASCII, so that there are no usable mono-spaced fonts. This means that consoles have had to be rewritten to cope with variable-width characters.

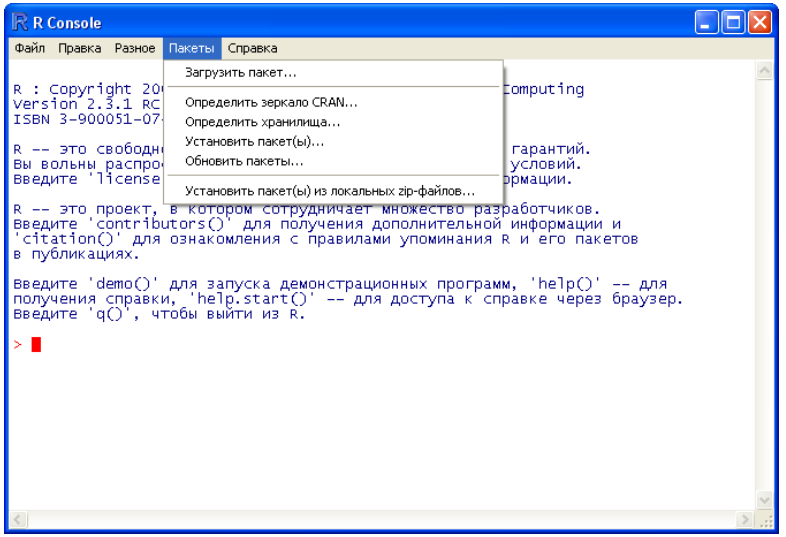
Localized RGui Consoles



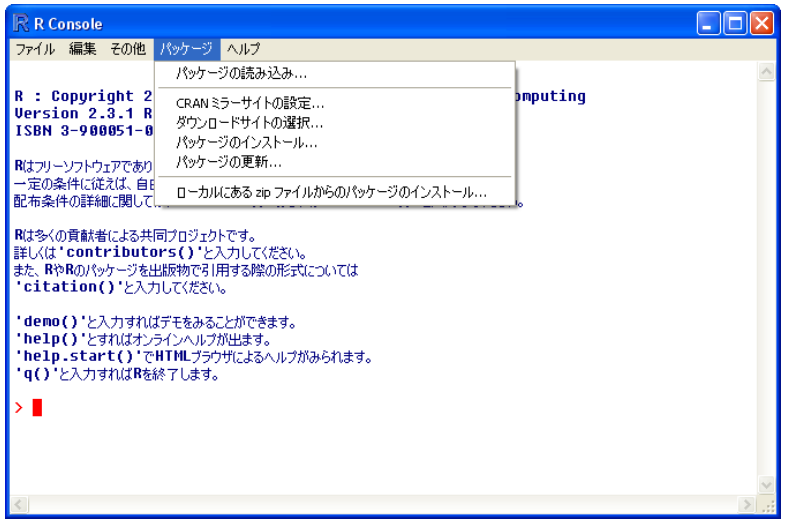
French



Russian



Japanese



Inputting Text

How does the user input text in a particular language? One way is to use a suitable keyboard, but even with virtual keyboards that can be tedious if several languages are involved. Sometimes it is possible to write separate documents in UTF-8 or UCS-2 and combine them in a suitable editor.

Unicode provides a number (the Unicode point) for all known characters. This is usually written in hex, e.g. U+201C (left double quotation mark). R 2.3.1 allows this to be entered in the form `\u201C` wherever the character is representable in the character set within which R is running.

Characters can always be input in a particular encoding as one or more bytes by octal or hex escapes, e.g. ń is `0xF1` in Latin 2.

Internationalization of Graphics Devices

Displaying non-ASCII glyphs is an issue for text output, but one we can leave to the OS.

For graphical output, it is an issue we have to address, especially for portable forms of graphics like PDF. R 2.1.0 provided support for the screen devices, provided suitable fonts are available. Ensuring that suitable fonts *are* available can be tricky, not least on X servers which do not give priority to finding the right encoding (essentially because there are very few Unicode-encoded fonts to choose from).

R 2.3.0 provided more support: e.g. on NT-based Windows Unicode output is used which enables you to output graphics in a different locale from that R is running in, and also to run R is a locale different from that R is running in.

PostScript and PDF devices

PostScript and PDF provided a considerable headache, one that Ei-ji Nakama, Paul Murrell and I worked on for a long time. The relatively easy part was to provide support for 8-bit encodings, and we have that for almost all Western and Eastern European languages, including Greek and Cyrillic (but only for one language group at a time). This can be extended by adding suitable coding files.

However, Adobe does not make use of Unicode, and the CJK languages are displayed in so-called CID fonts with private encodings. R 2.3.0 added support for CJK in a small range of easily-available CID fonts for `postscript` and `pdf`.

There are further issues: see the article by Paul Murrell and myself in the very latest R-News.

Portability Issues

The i18n of R allows you to write in your native language. Should you?

Probably not if your work is to be used outside a strictly controlled environment. Once you use any non-ASCII character, there is no guarantee that your users will see the character you intended, or any character at all! Which object names are syntactically valid depends on the locale.

It is possible to build R with no support for MBCS or i18n. It is not uncommon for commercial Unixen to have no support beyond Latin 1. Microsoft sells 'Non-international English' versions of Windows that have no support for changing language (it even sells them in Europe). And it is possible to run R in the `C` locale, which strictly only allows ASCII characters.

R enforces many of these restrictions on packages, assuming that they will be run in different environments. This is unfortunate for those who would like, e.g. to display a banner giving their authorship, but if they realize their names will most often be mangled they are likely to prefer a transliteration.

Conclusion

We started with

> Does R speak your language?

Conclusion

We started with

> Does R speak your language?

and the answer is

> R is capable of speaking your language.
>
> For most people it has already been taught how.

If you are one of those for which R only has potential, please consider contributing the localization needed.