MINED: An Editor with Extensive Unicode and CJK Support for the Text-based Terminal Environment

Thomas Wolff
http://towo.net/mined/
towo@computer.org

Introduction

Many Unicode editors are GUI applications that use a graphical or font-related API to display Unicode text.

But much computer work is done in traditional text-based terminal environments, like xterm.

MINED was the first editor that supported UTF-8 in this environment.

It tries to offer a most useful feature set for Unicode and CJK handling in the text mode terminal environment as the author believes that a non-graphic computer interface is still and will remain to be a flexible and for some people more convenient way of interaction and it deserves good support for modern features.

The paper, after a short overview of MINED general-purpose editing features, presents some specific Unicode and CJK support features and focuses on automatic and heuristic mechanisms intended to support the users smoothly with a variety of environment settings and use cases and relieve the users from configuration trouble.

MINED overview

MINED follows the design philosophy of being “simple and intuitive”.

That means, users should be enabled to perform the most frequent editing tasks without having to study and learn a peculiar interface first.

Unlike other editors that support plain text mode, MINED is affirmative toward the conveniences of modern user interface paradigms, like comprehensive menus, mouse control including mouse wheel scrolling, scrollbar navigation.

For character encoding support, MINED has the design philosophy of having everything “on board” so that e.g. CJK display and input methods run “out of the box” without any extra configuration and without depending on the operating system environment.

Text handling and editing support comprises

- Program editing features
- HTML support and syntax highlighting
- Identifier and function definition search, also across files
- Search and replacement patterns containing multiple lines

- Cross-session paste buffer (copy/paste between multiple, even subsequent or remote, invocations of MINED)
- Multiple paste buffers (emacs-style)
- Visible indications of special text contents (TAB characters, different line-end types, character codes that cannot be displayed in the current mode)
- Full binary transparent editing with visible indications (illegal UTF-8 or CJK, mixed line end types, NUL characters, ...);
  easy editing of files with mixed-encoding sections

Character encoding support and encoding auto-detection

MINED supports editing Unicode text (UTF-8 encoded), 8 bit encodings, as well as major CJK encodings (including GB18030 and full EUC-JP); CJK encoded files can be edited either in native CJK terminals or in Unicode terminals.

When loading a file, MINED has a heuristic approach to detect the encoding. It is very reliable for UTF-8 auto-detection. Distinguishing different CJK encodings works well in some cases; the user can configure the encodings to be considered for auto-detection to help the recognition.

Encoding can also be selected explicitly, or changed while editing text.

Typographic editing support

Smart quotes

In *smart quotes* mode, straight quote characters (double «"» or single «'») are automatically replaced with an opening or closing typographic quotation mark, depending on the text context.

A smart quotes style menu selects the kind of quote marks to be used, also there is some auto-detection implemented when a file is loaded by counting existing quote marks.

The smart quotes algorithm decides whether to insert a left quote or right quote depending on white space and punctuation context. In CJK text, this is not possible as white space is often not used. So if a CJK context is seen, a heuristic stateful approach is taken instead.

Smart dashes

If smart quotes are active, also an input sequence of «--» is replaced with an en dash (if preceded by a blank) or an em dash. A single «-» is replaced with a Hebrew hyphen mark Maqaf (U+05BE) if an adjacent character is in the Hebrew script range.

Indentation

In addition to *automatic indentation* (left margin alignment after newline) and back-tab (undent to previous indentation level) – two functions that are also very

useful for editing programs –, when paragraphs in bulleted lists are line-wrapped, they are automatically indented considering Unicode bullets and dashes.

Input support

MINED provides a number of mechanisms to support input of characters that are not directly available from the keyboard.

- *Mnemonic input support* covers character mnemonics from RFC 1345, HTML mnemos, TeX mnemos (macros) and substitutes, and additional mnemonics that appear to be useful. The latter also complete the generic mnemonic notations for accented character from RFC 1345 to the full Unicode range of characters using such accents.
- Numeric input support (compliant with ISO 14755).
- Function keys used as *accent prefixes* (like "dead keys") for the most frequent accent combinations.
- Specific accent prefix extensions for Vietnamese multiple accent combinations.
- *Keyboard mappings* that remap the keyboard for efficient text entry in non-Latin languages. By default, MINED is equipped with keyboard mappings for Greek, Cyrillic, Hebrew, Arabic, and with mappings that provide input methods for East Asian languages.

Input methods

Especially for East Asian *input methods*, keyboard mapping alone is not sufficient. In many cases a sequence of input keys is mapped to a range of different letters/symbols or words.

In that case a character selection menu or *"pick list"* is opened in which the user can navigate and choose with keyboard or mouse control. Characters in the pick list are sorted by relevance of Unicode ranges.

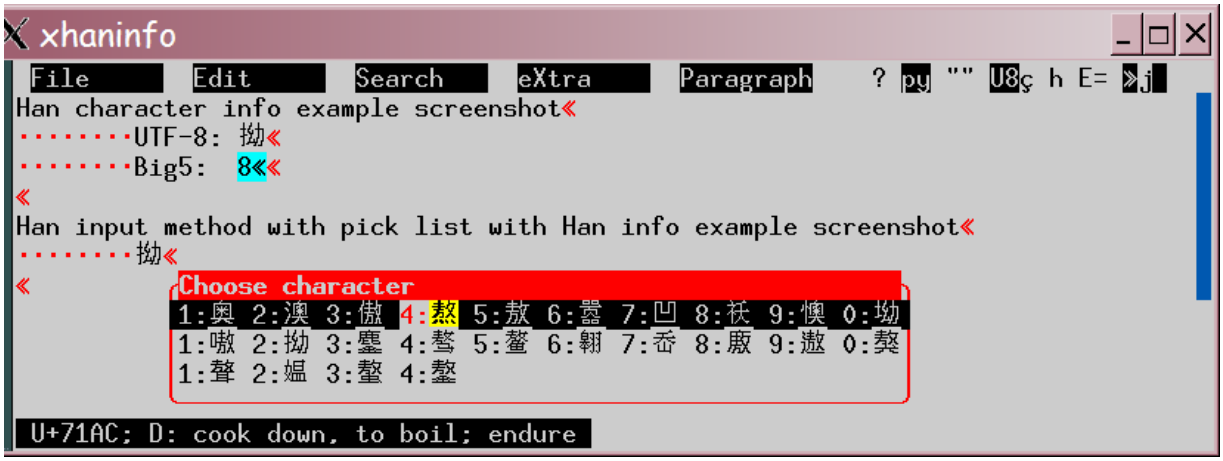

Figure 1: Pick list offering multiple choices for mapped keyboard input

A special mechanism is provided for the Radical/Stroke input method which provides a means of entering CJK characters for unexperienced users. It follows a two-level approach, first letting the user select one of the 214 CJK "radicals", then offering a pick list for the symbols based on the chosen radical, sorted by the number of additional strokes.

MINED input methods work with built-in tables, providing the users a portable way of entering characters that is always available without additional configuration or setup of software in the system environment.

While certain other input methods may have some more comfortable features, especially for Japanese where intelligent transformations are desired, the MINED approach has advantages for quick and portable character input, even on legacy systems.

Character handling features

Separated display mode

For combined characters (consisting of a Unicode base character and one or more combining characters), a *separated display mode* is available. It displays combining characters separately and highlights them for easy recognition.

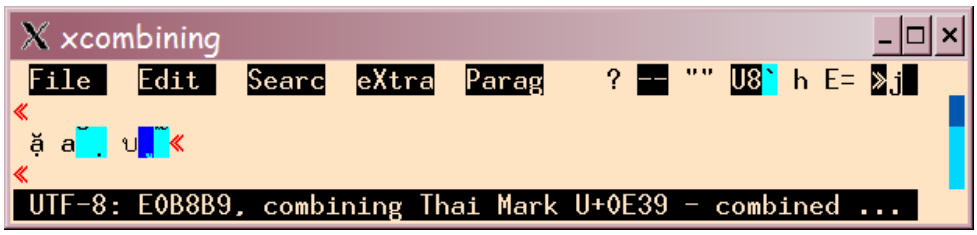

Figure 2: Viewing and navigating combining characters in separated display mode

Combined editing

In *combined display mode* (the default), there are functions to navigate into and partially edit a combined character (Control-Left/Right for "micro movement" within a displayed combined character, Control-Backspace to delete only the last combining character). Also the character information display supports recognising the character structure of the text if needed.

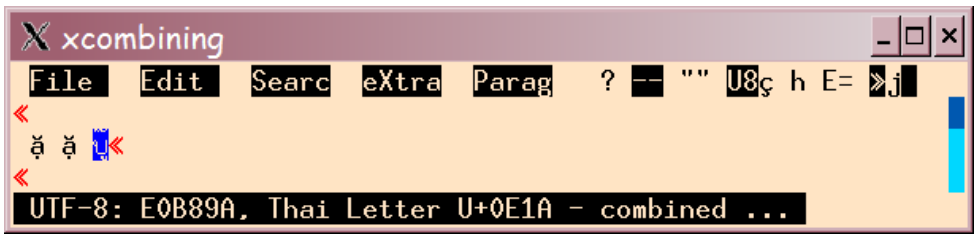

Figure 3: Combined display: character information on base character

Figure 4: Combined editing: positioning into the combined character

Script highlighting

To distinguish similar glyphs of different scripts (esp. Greek and Cyrillic versus Latin), *script highlighting* indicates Unicode script ranges by using different colours.

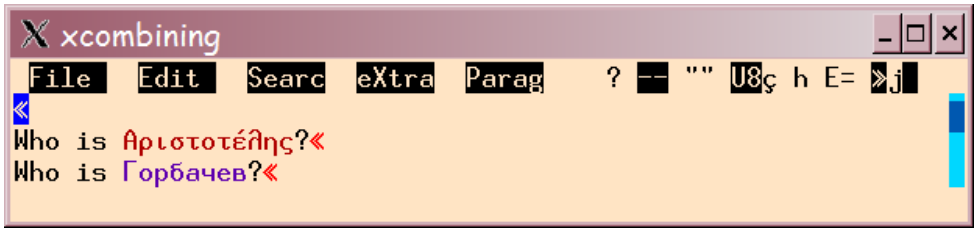

Figure 5: Script highlighting, indicating Greek and Cyrillic script ranges

Han character information display

For Han characters (CJK text), a *Han information display* mode can be activated that shows semantic and pronunciation information about the character under the cursor position. Information is derived from the Unihan database (with some automatic typographic fixes).

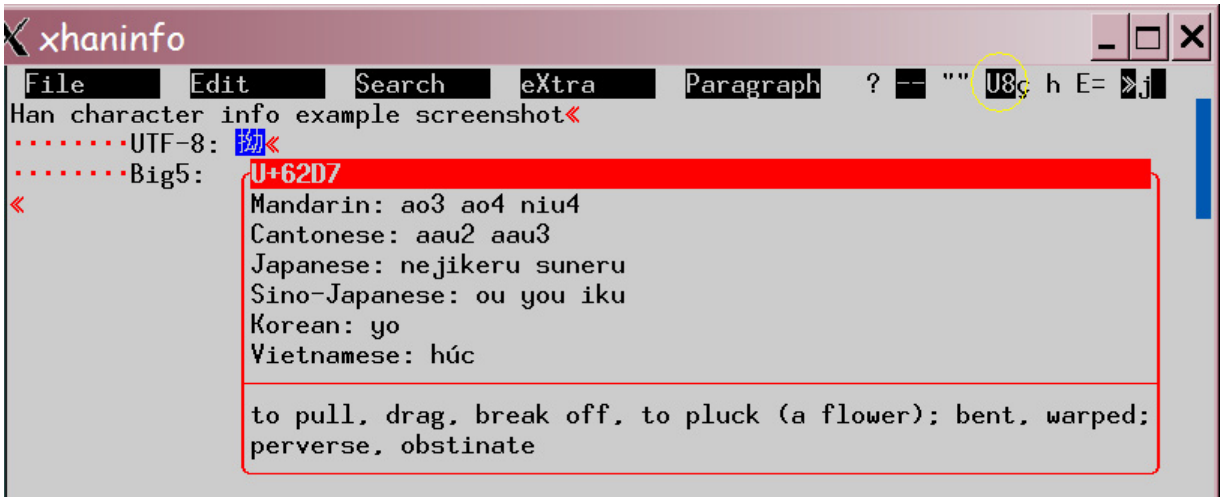

Figure 6: Han information display, showing all available CJK character information

This information is also displayed with an input method pick list to help selecting a suitable character.

Support of future Unicode data versions

The MINED makefiles and building scripts include automatic generation of character handling information tables from updated Unicode data files.

Interactive conversion functions

Case toggle and similar functionality

The *case toggle* function handles special cases like Greek final sigma and Turkish i (the latter depending on locale variable settings).

As an additional functionality, it also exchanges corresponding Hiragana/Katakana.

Another function helps with interactive conversion between Latin-1 and UTF-8 encoding, in case this is desired for a part of the text only.

Mnemonic conversion

A function is available to transform a mnemonic embedded in the text into its associated character; this may be useful, e.g., to fix text that has been written in pure ASCII, using dreary replacements of accented letters. In addition to the complete set of mnemonics also used for mnemonic input support, a variation of commands is available to perform this function with national preferences:

ESC ä entered on text oe replaces it with ö

ESC ç entered on text oe replaces it with œ

ESC å entered on text oe replaces it with ø

The reason behind this is that these command letters are available on the respective national keyboards, so it's intuitive to use them for such a preference transformation.

ESC _ entered on text oe replaces it with one of the above, depending on locale variable settings.

Terminal feature auto-detection

As a problem with the text terminal environment, there is no well-defined uniform interface for character properties to make their handling compliant with that of the terminal. Most editors cannot handle display of wide versus single-width characters and combining characters consistently for a range of differently behaving terminals.

The locale mechanism is not a sufficient remedy here for a number of reasons.

Why does the locale mechanism fail to support the terminal?

The intention of the locale mechanism is to maintain a data base about character properties (here esp. width information) for use by applications.

But it cannot reliably support text-mode terminal operation due to a number of problems:

- Trouble # 1: The locale mechanism demands the users to undergo hassles of configuration trouble; an ordinary user is already overcharged with finding out whether to set e.g. LC_CTYPE=de.UTF-8 or de_DE.utf8 because variations occur on various systems (especially in a heterogeneous network), and the shell profile may have to support a mapping from hostnames to working locale values. Moreover, the locale value combines various information into one setting: language, regional variations, and character encoding. While the use case would actually be to set these independently, some environments are quite strict about demanding the exact match of the configured locale value to an installed locale. So if a system has only en_US.UTF-8 installed as a UTF-8 locale but the users set de_something.UTF-8 because that's what they want, the system will complain and software relying on its locale mechanism will fail. While there may be minor implementation reasons for this strictness, it imposes serious limitations on the flexibility the users are allowed for locale configuration, and the mechanism should be improved to provide a more generic setting of encoding configuration, derived from the locale variables.
- If a user finally finds out the desired locale is e.g. fr_FR.UTF-8 but it's not installed on the system, the user is often stuck.
  Documentation about locale installation is poor and system administrators are often reluctant about support for this feature they don't care about. Also not all system have locale support yet.
- Trouble # 2: If the user is lucky to find a suitable locale installed on the used systems and is told which value(s) to set to address it, this doesn't mean that the desired tools use it!
  E.g. xterm maintains its own character width tables while rxvt refers to locale information to drive the screen.
- Trouble # 3: Different versions of character data (referring to different versions of Unicode) may be used by various applications (e.g. terminal emulators) and may even be installed as locales on different machines in a heterogeneous network.
  So if you are lucky to have found a working configuration on one machine, you should expect that it doesn't work anymore after a remote login / telnet. Even within one system, there is no guarantee or even approximation of uniform behaviour – e.g. the Linux console does not support double-width or combining characters while xterm does; but there are no different locales installed to accommodate this and if there were we would be back at trouble # 1.

Handling character properties of different terminals

The approach of MINED is to detect terminal properties automatically and to adjust its handling of character cell width and characters that are combined, to ensure maximal display consistency for the users.

This is achieved with an auto-detection mechanism that determines the width of a selected set of test strings and derives the according properties. The width is

determined by means of the cursor position report of modern terminals (with a time-out for terminals that don't respond to this request).

The first and major test string combines a number of basic properties, and some special cases are probed then, some of them conditionally. The following table shows some of the test strings and what information they give, providing an overview about the mechanism and how it works; the test strings are shown here in their Unicode representation, they are actually byte sequences and would have different meanings in different encodings.

String	**Width**	**Conclusion**
åلاឃឃ 刈墢	6	UTF-8 terminal, no double-width support, with Arabic LAM/ALEF ligature joining
	7	UTF-8, no double-width, no LAM/ALEF ligature joining
	8	UTF-8, double-width, with LAM/ALEF ligature joining (probably mlterm)
	9	UTF-8, double-width, no LAM/ALEF ligature joining
	10, 11, 14, 16, 17	CJK encoded terminal
	15, 18	8 bit terminal or CJK terminal
a U+0321	1	terminal supports combining characters
	2	terminal does not support combining characters
〚〛｠ U+301A U+301B U+FF60	< 8	xterm version 157–166, corresponding Unicode version 3.2
‘’“”…—	> 8	xterm with option -cjk_width
0x8130A132 («Ǧ» encoded in GB18030)	> 2	not a GB18030 terminal
0xA1A4A1B1EAA5A6A1	< 10	a non-native CJK terminal, using Unicode width data (e.g. luit)

In addition to this mechanism, to set the encoding for CJK terminals, or if the terminal doesn't report cursor positions, information from the locale environment

variables (LC_ALL, LC_CTYPE, LANG) is used to determine the user's configuration settings (not using the system locale functionality however).

Conclusion

MINED implements character encoding in a way that should work "out of the box" for most environments and thus helps its users to work with Unicode text (as well as CJK) without any configuration needs. Its encoding handling and auto-detection has been tested with xterm, mlterm, hanterm, cxterm, rxvt, linux console.

MINED brings this together with a range of text and program editing features, an intuitive user interface, and a robust text and file handling engine.

It compiles and runs on many platforms, including legacy systems: Unix (Linux/Sun/HP/BSD/Mac and more), DOS (djgpp), Windows (cygwin).

With MINED, a new version of xterm, and some font setup, you can equip old systems (that have even never heard of locales) with Unicode support.

Together with its "small-footprint" behaviour (quick start-up, no libraries and plug-ins to be loaded) it should be a suitable tool providing useful features for a number of users.

About the Author

Thomas Wolff studied computer science with a mathematical background, received a doctoral degree in object-orientation and distribution.

Major interests are in programming languages and concurrency.

Other interests are in design (typography, user interfaces, usability).

Working as a systems engineer for telecommunication servers.