JCBmacs Documentation

This is the documentation for JCBmacs (thank you, Aidan Kehoe), which is a fork of XEmacs 21.4, using Unicode as its main internal character representation. (As with recent GNU Emacs, it can also represent all the traditional character sets, independently from Unicode. It is therefore highly backward-compatible.)

This documentation is still cursory, and is split into three parts: stuff the non-programming user needs to know, stuff the Lisp programmer needs to know, and stuff the C programmer needs to know.

1 User-level changes

There are, intentionally, very few changes immediately obvious in a vanilla JCBmacs. The most obvious is that if you visit a file that is in utf-8, JCBmacs will recognize it, and the symbol 'U8' will appear at the left of the modeline in the coding-system slot. Also, of course, `utf-8` is available as a coding system for explicit choice. (So should the other Unicode coding systems, but for no reason other than laziness, they aren't there yet.)

If you have been used to using the `mule-ucs` package for Unicode support, you should not try to use it with JCBmacs. An appropriate snippet in your init file is:

```
(if (fboundp 'unicode-precedence-list)
    t
  (require 'un-define)
  (require 'unicode)
  (set-coding-category-system 'utf-8 'utf-8))
```

which works because both JCBmacs and recent XEmacs 21.5 define `unicode-precedence-list`.

For compatibility, JCBmacs retains the traditional XEmacs default coding system of ISO-2022. Assuming you are in the modern world, you will want to change this to utf-8, which you can do with the following entry in your '`.xemacs/init.el`' file:

```
(setq-default buffer-file-coding-system 'utf-8)
```

You will probably also want to make utf-8 the highest priority coding system, which can be done by

```
(set-coding-priority-list (cons 'utf-8 (delq 'utf-8 (coding-priority-list))))
```

When editing, all the usual XEmacs input methods still work. Characters are inserted in their native character sets, so, for example, Big5 and GB2312 characters are distinct. As yet, there is no automatic folding of characters into UCS in the Emacs buffer. If the buffer is in the utf-8 coding system, all characters will be converted to Unicode on save. Thus, the simplest method to convert the buffer to Unicode is to save the file and then revert the buffer.

Any Unicode character may be inserted with the command `insert-ucs-character`, which is bound to *C-x U* (note the upper-case *U*). Interactively, a UCS code point *in hexadecimal* is read from the minibuffer.

For choosing fonts, JCBmacs allows flexibility in using legacy and Unicode fonts. Its behaviour is controlled by the (awkwardly named) variable `display-fonts-per-character`. If you have Unicode fonts but not legacy fonts, you will probably want to set this to 3. If you have plenty of legacy fonts, but not good Unicode coverage, set it to 2. The default value of 1 is appropriate for systems with good font coverage of both legacy and Unicode characters.

JCBmacs implements `visual-line-mode`, similar to that in GNU Emacs.

2 Lisp-level changes

This chapter describes the changes visible at the Lisp level. It follows the structure of the Lisp Reference Manual.

2.1 Lisp Data Types

The following primitive types have changed:

2.1.1 Characters

The integer values of characters (above 255) have changed. For UCS values, the integer value is the UCS codepoint. For legacy characters, the integer value is not useful. Character (and string) read syntax is extended with \x and \u escapes: these may be followed by up to 8 hexadecimal digits (for compatibility with Gnu Emacs syntax), but will give an error if the resulting integer is not a valid character (for \x) or is not a Unicode character (for \u). Unlike GNU Emacs, it is not possible to follow the digits with '\ ' (escaped space) to terminate the digit string; if you need to follow a \u or \x by a hex digit character, you must either use a full eight digits, or terminate with an escaped newline.

2.1.2 Char tables

Char tables have been extended as follows:

The types of ranges that can be assigned values are, for Mule characters:

all Mule and Unicode characters (represented by t)

all Mule characters (represented by 'mule)

an entire charset

a single row in a two-octet charset (represented by a vector of two elements: a two-octet charset and a row number; the row must be an integer, not a character)

a single character

For Unicode characters, the ranges are

all UCS characters (represented by 'ucs)

a 14-bit segment of UCS (represented by a vector of two elements: the starting point of the segment, and the size 0x4000 of the segment);

a 7-bit segment of UCS (represented by a vector of two elements: the starting point of the segment, and the size 0x80);

a single character

Display tables have changed similarly.

2.1.3 Charsets and Coding Systems

These have changed. See MULE Section 2.7.1 [Charsets], page 7 and MULE Section 2.7.4 [Coding Systems], page 8

2.2 Byte Compilation

Warning: JCBmacs byte-compiled files may not be understood by XEmacs 21.4, because the esc-quoted coding system used for byte-compiled files has been extended to allow Unicode characters.

2.3 Read and Print

One of the major gotchas of XEmacs is that if data is corrupted because of a wrong choice of coding system during i/o, you get no warning.

It is technically infeasible to return an error, as en-/de-coding happens in a place far removed from the function doing the i/o. However, JCBmacs will throw up a warning (in the *Warnings* buffer) whenever invalid input is read, or a character cannot be written to output in the current coding system. Such warnings should always be investigated. (The VM mail reader used to have many such errors, leading to major data corruption for multilingual users.)

2.4 Searching and Matching

Unfortunately no change has been made to the searching and matching functionality, except to disable 'fastmap' optimization, which is too hard to support. (It doesn't seem to matter, though.)

What ought to happen is: implementation of standard modern Unicode regexp constructs, and 'folding' of character sets so that equivalent characters are not distinguished.

More urgently, incremental search needs to work with input methods. This, alas, seems to be a hard problem. GNU Emacs has it working, but their input methods are lower level. It's probably the only way.

2.5 Fonts

In XEmacs, fonts are usually associated with charsets; in traditional X fonts, the last part of the XLFD identifies the charset (not quite, but close enough), and the first matching font is used to display any character in the charset. A consequence of this is that each charset needs complete coverage in its font. This doesn't work too well with Unicode, as most fonts have partial coverage, e.g. simplified Chinese, or Greek, or Armenian.

Therefore JCBmacs, by default, looks for fonts by the individual character, and chooses the first matching font that actually contains a glyph for the character. Optionally, it can also fold legacy characters into Unicode glyphs, and vice versa. This process is controlled by the variable `display-fonts-per-character`, whose documentation gives more details.

As part of handling such matters, and to provide basic handling of combining characters (diacritics etc.), various display properties are defined per character. This will ultimately be the Unicode property database, but at present it's a chartab, which is not exposed to Lisp. Currently defined properties are individually exported by functions `char-display-get-*` and `char-display-set-*`. This interface is likely to change.

The traditional X Window font system predates multi-plane Unicode, and limits fonts to 65536 glyphs. Therefore, JCBmacs makes the technically unfounded but pragmatically

reasonable decision to steal the CHARSET_ENCODING field of the XLFD for Unicode fonts to give the plane. So a `*-iso10646-1` font is assumed to be plane 1, and a plane 2 font can be named `*-iso10646-2`, and so on.

2.6 Emacs Display

For no reason other than that I wanted it, JCBmacs implements a 'long lines mode', which is actually called `visual-line-mode`. The documentation is in the function. There is currently a bug in the cursor handling, which leaves ghost cursors at the ends of lines.

2.7 MULE

2.7.1 Charsets

The term *legacy charset* means the traditional XEmacs charsets, i.e. those other than Unicode.

XEmacs charsets are (apart from one or two special cases) either 94, 96, 94x94 or 96x96. JCBmacs adds the UCS charset, which is a one-dimensional charset with 1114112 characters, and, for internal use, charsets for each plane of the UCS, which are 256x256 charsets.

There are additions and changes to the properties that charsets have, as follows:

`registry` See Section 2.5 [Fonts], page 6 for how JCBmacs (ab)uses the registry for Unicode fonts.

`to_ucs` This property specifies a chartab for converting characters in this charset to UCS. If it is `nil`, the chartab stored in `mule-to-ucs` is used (this chartab is generated from mapping tables by Lisp initialization code).

`from_ucs` This property is a chartab for conversion from UCS to this chartab. It will be filled in by Lisp initialization code from mapping tables. This chartab should not be shared between different charsets. Conversion from UCS to legacy uses the chartab in the global variable `ucs-to-mule`, which is constructed from `from_ucs` entries in the user's preferred priority order.

`shadow` A shadow charset is a mirror of a real charset, but may have different name, registry and direction properties. `shadow` is the real charset that this charset is shadowing. This was introduced for allowing more control of fonts, but is probably unnecessary, evil and to be deleted.

`chars` This property may additionally have the values 1114112 or 256.

The following new basic charset function exists (but may be removed):

make-shadow-charset *charset-or-name name doc-string props* Function

This function makes a charset shadowing an existing charset. *charset-or-name* is the charset to be shadowed. The other arguments are as for `make-charset`. Only the `short-name`, `long-name`, `registry` and `direction` properties are recognized.

The following additional charsets are predefined in the C code:

```
Name                     Type  Fi Gr Dir Registry
-------------------------------------------------------------
ucs                   1114112       0  l2r ISO10646
ucs-plane-0           256x256       0  l2r ISO10646
 ...
ucs-plane-16          256x256       0  l2r ISO10646
```

The UCS plane charsets are for internal use, and may be removed in future, or at least hidden from lisp. The `l2r` and `graphic` properties are not meaningful.

The `composite` charset is not defined.

The `ethiopic` and `ascii-r2l` charsets are no longer defined in Lisp.

2.7.2 MULE Characters

If the `make-char` function is called with the `ucs` charset, *arg1* may be any UCS codepoint. `make-char` with a shadow or UCS plane charset is not supported.

Similarly, `char-octet` returns the codepoint of a UCS character.

2.7.3 Composite Characters

All support for composite characters has been removed.

2.7.4 Coding Systems

There is a `utf-16` coding system type, and a `gb18030` type. The `ucs-4` and `utf-16` types recognize the following additional properties:

`little-endian`
: Non-nil if little-endian format is used; otherwise big-endian.

`bom`
: If this is nil, no byte order mark is accepted or generated. If it is t, a byte order mark is accepted on input and generated on output. If it is any other value, a byte order mark is accepted on input, but not generated on output.

The Chinese `gb18030` coding system is predefined in C. Since it is defined to be a mapping of Unicode, characters read with it are in the UCS charset, not in the legacy Chinese charsets, even when they are part of the legacy charsets.

The `escape-quoted` coding system has been extended to use the ISO2022 Unicode escape sequence for UCS characters.

All built-in coding systems will throw a warning if they encounter an error: undecodable bytes on decoding, or an unencodable character on encoding.

Functions relevant to conversion between legacy charsets and UCS are:

set-ucs-char *codepoint character* Function
: makes *character* the preferred legacy version of the UCS *codepoint*

ucs-char *codepoint* Function
: returns the preferred legacy version of the UCS *codepoint*, or `nil` if not set

set-char-ucs *character codepoint* Function
makes *character* map to UCS *codepoint*, both in the `from-ucs` property of *character*'s charset and in the global `mule-to-ucs` chartab.

char-ucs *character* Function
returns the UCS codepoint corresponding to *character*, or `nil` if not set.

2.7.5 Input Methods

In order that input methods can work in isearch and other such places, they have been reimplemented along the same lines as GNU Emacs, and '`isearch-mode.el`' adjusted accordingly. The implementation is a bit hacky.

In '`event-mod.h`', a fake modifier bit `XEMACS_MOD_INPUT` is added, which allows events to be labelled as coming from an input method rather than from the outside. It has the Lisp name `input`, and is ignored by keymap lookup. Input methods store their output in a lisp variable, whence they are processed by the main event stream in priority to external events.

The following variables control input methods:

input-method-function Variable
If this function is defined, a keypress event is passed to this function instead of being dispatched in the normal way. The function should return an event or list of events to be processed in place of the original event. Such events are dispatched normally, ahead of all other pending command events. This variable is reset to `nil` if any error occurs in the execution of the function.

unread-post-input-method-events Variable
Variable containing the list of events from an input method. These events are dispatched ahead of all other event sources, and are never given to an input method. Input methods should (for FSF compatibility) simply return events, rather than directly adding them to this list.

input-method-previous-message Variable
Variable containing previous contents of echo area (before current event).

Instead of the '`quail.el`' provided with XEmacs 21.4, a replacement file is preloaded. This file was taken from the last GPL2 GNU Emacs, and then adapted for the input method implementatin used here. The adaptation is not complete: keyboard translation is not implemented (does anybody use it?), and the conversion needed for Japanese input methods is not implemented.

2.7.6 CCL

CCL should be obsolete, but backward compatibility requires it to be supported. Since CCL has access to the internal representation of XEmacs characters and buffers, this is a problem. JCBmacs solves this problem by assuming that all coding system CCL is legacy, and translating between JCBmacs and XEmacs representations when invoking CCL coding

systems. A corollary of this is that CCL coding systems cannot access UCS characters. If anyone ever wished to use CCL in the context of UCS, it would be easy to add a new property to coding systems to disable the representation conversion.

Sample index entry 3

Table of Contents

1 User-level changes 3

2 Lisp-level changes 5

2.1 Lisp Data Types 5
2.1.1 Characters 5
2.1.2 Char tables 5
2.1.3 Charsets and Coding Systems 5
2.2 Byte Compilation 6
2.3 Read and Print 6
2.4 Searching and Matching 6
2.5 Fonts 6
2.6 Emacs Display 7
2.7 MULE 7
2.7.1 Charsets 7
2.7.2 MULE Characters 8
2.7.3 Composite Characters 8
2.7.4 Coding Systems 8
2.7.5 Input Methods 9
2.7.6 CCL 9