## NAME

Encode::Locale - Determine the locale encoding

## SYNOPSIS

```
use Encode::Locale;
use Encode;

$string = decode(locale => $bytes);
$bytes = encode(locale => $string);

if (-t) {
binmode(STDIN, ":encoding(console_in)");
binmode(STDOUT, ":encoding(console_out)");
binmode(STDERR, ":encoding(console_out)");
}

# Processing file names passed in as arguments
my $uni_filename = decode(locale => $ARGV[0]);
open(my $fh, "<", encode(locale_fs => $uni_filename))
|| die "Can't open '$uni_filename': $!";
binmode($fh, ":encoding(locale)");
...
```

## DESCRIPTION

In many applications it's wise to let Perl use Unicode for the strings it processes. Most of the interfaces Perl has to the outside world are still byte based. Programs therefore need to decode byte strings that enter the program from the outside and encode them again on the way out.

The POSIX locale system is used to specify both the language conventions requested by the user and the preferred character set to consume and output. The `Encode::Locale` module looks up the charset and encoding (called a CODESET in the locale jargon) and arranges for the Encode module to know this encoding under the name "locale". It means bytes obtained from the environment can be converted to Unicode strings by calling `Encode::encode(locale => $bytes)` and converted back again with `Encode::decode(locale => $string)`.

Where file systems interfaces pass file names in and out of the program we also need care. The trend is for operating systems to use a fixed file encoding that don't actually depend on the locale; and this module determines the most appropriate encoding for file names. The Encode module will know this encoding under the name "locale_fs". For traditional Unix systems this will be an alias to the same encoding as "locale".

For programs running in a terminal window (called a "Console" on some systems) the "locale" encoding is usually a good choice for what to expect as input and output. Some systems allows us to query the encoding set for the terminal and `Encode::Locale` will do that if available and make these encodings known under the `Encode` aliases "console_in" and "console_out". For systems where we can't determine the terminal encoding these will be aliased as the same encoding as "locale". The advice is to use "console_in" for input known to come from the terminal and "console_out" for output known to go from the terminal.

In addition to arranging for various Encode aliases the following functions and variables are provided:

decode_argv( )
decode_argv( Encode::FB_CROAK )

This will decode the command line arguments to perl (the `@ARGV` array) in-place.

The function will by default replace characters that can't be decoded by "x{FFFD}", the Unicode replacement character.

Any argument provided is passed as CHECK to underlying *Encode::decode()* call. Pass the

value `Encode::FB_CROAK` to have the decoding croak if not all the command line arguments can be decoded. See ''Handling Malformed Data'' in Encode for details on other options for CHECK.

env( `$uni_key` )
env( `$uni_key` => `$uni_value` )
Interface to get/set environment variables. Returns the current value as a Unicode string. The `$uni_key` and `$uni_value` arguments are expected to be Unicode strings as well. Passing `undef` as `$uni_value` deletes the environment variable named `$uni_key`.

The returned value will have the characters that can't be decoded replaced by ''x{FFFD}'', the Unicode replacement character.

There is no interface to request alternative CHECK behavior as for *decode_argv()*. If you need that you need to call encode/decode yourself. For example:

```
my $key = Encode::encode(locale => $uni_key, Encode::FB_CROAK);
my $uni_value = Encode::decode(locale => $ENV{$key}, Encode::FB_CROAK);
```

reinit( )
reinit( `$encoding` )
Reinitialize the encodings from the locale. You want to call this function if you changed anything in the environment that might influence the locale.

This function will croak if the determined encoding isn't recognized by the Encode module.

With argument force `$ENCODING_`... variables to set to the given value.

`$ENCODING_LOCALE`
The encoding name determined to be suitable for the current locale. Encode know this encoding as ''locale''.

`$ENCODING_LOCALE_FS`
The encoding name determined to be suiteable for file system interfaces involving file names. Encode know this encoding as ''locale_fs''.

`$ENCODING_CONSOLE_IN`
`$ENCODING_CONSOLE_OUT`
The encodings to be used for reading and writing output to the a console. Encode know these encodings as ''console_in'' and ''console_out''.

## NOTES

This table summarizes the mapping of the encodings set up by the `Encode::Locale` module:

```
Encode | | |
Alias | Windows | Mac OS X | POSIX
-----------+---------+--------------+------------
locale | ANSI | nl_langinfo | nl_langinfo
locale_fs | ANSI | UTF-8 | nl_langinfo
console_in | OEM | nl_langinfo | nl_langinfo
console_out | OEM | nl_langinfo | nl_langinfo
```

### Windows

Windows has basically 2 sets of APIs. A wide API (based on passing UTF-16 strings) and a byte based API based a character set called ANSI. The regular Perl interfaces to the OS currently only uses the ANSI APIs. Unfortunately ANSI is not a single character set.

The encoding that corresponds to ANSI varies between different editions of Windows. For many western editions of Windows ANSI corresponds to CP-1252 which is a character set similar to ISO-8859-1. Conceptually the ANSI character set is a similar concept to the POSIX locale CODESET so this module figures out what the ANSI code page is and make this available as `$ENCODING_LOCALE` and the ''locale'' Encoding alias.

Windows systems also operate with another byte based character set. It's called the OEM code page. This is the encoding that the Console takes as input and output. It's common for the OEM code page to differ from the ANSI code page.

### Mac OS X

On Mac OS X the file system encoding is always UTF-8 while the locale can otherwise be set up as normal for POSIX systems.

File names on Mac OS X will at the OS-level be converted to NFD-form. A file created by passing a NFC-filename will come in NFD-form from *readdir()*. See Unicode::Normalize for details of NFD/NFC.

Actually, Apple does not follow the Unicode NFD standard since not all character ranges are decomposed. The claim is that this avoids problems with round trip conversions from old Mac text encodings. See Encode::UTF8Mac for details.

### POSIX (Linux and other Unixes)

File systems might vary in what encoding is to be used for filenames. Since this module has no way to actually figure out what the is correct it goes with the best guess which is to assume filenames are encoding according to the current locale. Users are advised to always specify UTF-8 as the locale charset.

## SEE ALSO

I18N::Langinfo, Encode

## AUTHOR