NAME

Locale::Recode − Object−Oriented Portable Charset Conversion

SYNOPSIS

```
use Locale::Recode;

$cd = Locale::Recode->new (from => 'UTF-8',
                           to   => 'ISO-8859-1');

die $cd->getError if $cd->getError;

$cd->recode ($text) or die $cd->getError;

$mime_name = Locale::Recode->resolveAlias ('latin-1');

$supported = Locale::Recode->getSupported;

$complete = Locale::Recode->getCharsets;
```

DESCRIPTION

This module provides routines that convert textual data from one codeset to another in a portable way. The module has been started before **Encode** (3) was written. It's main purpose today is to provide charset conversion even when **Encode** (3) is not available on the system. It should also work for older Perl versions without Unicode support.

Internally **Locale::Recode** (3) will use **Encode** (3) whenever possible, to allow for a faster conversion and for a wider range of supported charsets, and will only fall back to the Perl implementation when **Encode** (3) is not available or does not support a particular charset that **Locale::Recode** (3) does.

Locale::Recode (3) is part of libintl-perl, and it's main purpose is actually to implement a portable charset conversion framework for the message translation facilities described in **Locale::TextDomain** (3).

CONSTRUCTOR

The constructor `new()` requires two named arguments:

from
: The encoding of the original data. Case doesn't matter, aliases are resolved.

to The target encoding. Again, case doesn't matter, and aliases are resolved.

The constructor will never fail. In case of an error, the object's internal state is set to bad and it will refuse to do any conversions. You can inquire the reason for the failure with the method **getError()**.

OBJECT METHODS

The following object methods are available.

recode (STRING)
: Converts **STRING** from the source encoding into the destination encoding. In case of success, a truth value is returned, false otherwise. You can inquire the reason for the failure with the method **getError()**.

getError
: Returns either false if the object is not in an error state or an error message.

CLASS METHODS

The object provides some additional class methods:

getSupported
: Returns a reference to a list of all supported charsets. This may implicitly load additional **Encode** (3) conversions like **Encode::HanExtra** (3) which may produce considerable load on your system.

  The method is therefore not intended for regular use but rather for getting resp. displaying *once* a list of available encodings.

The members of the list are all converted to uppercase!

getCharsets

Like **getSupported()** but also returns all available aliases.

SUPPORTED CHARSETS

The range of supported charsets is system-dependent. The following somewhat special charsets are always available:

UTF−8

UTF−8 is available independently of your Perl version. For Perl 5.6 or better or in the presence of **Encode**(3), conversions are not done in Perl but with the interfaces provided by these facilities which are written in C, hence much faster.

Encoding data *into* UTF−8 is fast, even if it is done in Perl. Decoding it in Perl may become quite slow. If you frequently have to decode UTF−8 with **Locale::Recode** you will probably want to make sure that you do that with Perl 5.6 or beter, or install **Encode**(3) to speed up things.

INTERNAL

UTF−8 is fast to write but hard to read for applications. It is therefore not the worst for internal string representation but not far from that. **Locale::Recode**(3) stores strings internally as a reference to an array of integer values like most programming languages (Perl is an exception) do, trading memory for performance.

The integer values are the UCS−4 codes of the characters in host byte order.

The encoding **INTERNAL** is directly available via **Locale::Recode**(3) but of course you should not really use it for data exchange, unless you know what you are doing.

Locale::Recode(3) has native support for a plethora of other encodings, most of them 8 bit encodings that are fast to decode, including most encodings used on popular micros like the ISO−8859−* series of encodings, most Windows−* encodings (also known as CP*), Macintosh, Atari, etc.

NAMES AND ALIASES

Each charset resp. encoding is available internally under a unique name. Whenever the information was available, the preferred MIME name (see <http://www.iana.org/assignments/character−sets/>) was chosen as the internal name.

Alias handling is quite strict. The module does not make wild guesses at what you mean (‘‘What’s the meaning of the acronym JIS’’ is a valid alias for ‘‘7bit−jis’’ in **Encode**(3) ....) but aims at providing common aliases only. The same applies to so-called aliases that are really mistakes, like ‘‘utf8’’ for UTF−8.

The module knows all aliases that are listed with the IANA character set registry (<http://www.iana.org/assignments/character−sets/>), plus those known to libiconv version 1.8, and a bunch of additional ones.

CONVERSION TABLES

The conversion tables have either been taken from official sources like the IANA or the Unicode Consortium, from Bruno Haible’s libiconv, or from the sources of the GNU libc and the regression tests for libintl-perl will check for conformance here. For some encodings this data differs from **Encode**(3)’s data which would cause these tests to fail. In these cases, the module will not invoke the **Encode**(3) methods, but will fall back to the internal implementation for the sake of consistency.

The few encodings that are affected are so simple that you will not experience any real performance penalty unless you convert large chunks of data. But the package is not really intended for such use anyway, and since **Encode**(3) is relatively new, I rather think that the differences are bugs in Encode which will be fixed soon.

BUGS

The module should provide fall back conversions for other Unicode encoding schemes like UCS−2, UCS−4 (big− and little-endian).

The pure Perl UTF−8 decoder will not always handle corrupt UTF−8 correctly, especially at the end and at

the beginning of the string. This is not likely to be fixed, since the module's intention is not to be a consistency checker for UTF−8 data.

AUTHOR

SEE ALSO

Encode (3), **iconv** (3), **iconv** (1), **recode** (1), **perl** (1)