NAME

Encode - character encodings

SYNOPSIS

use Encode;

Table of Contents

Encode consists of a collection of modules whose details are too big to fit in one document. This POD itself explains the top-level APIs and general topics at a glance. For other topics and more details, see the PODs below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encode::Alias</td>
<td>Alias definitions to encodings</td>
</tr>
<tr>
<td>Encode::Encoding</td>
<td>Encode Implementation Base Class</td>
</tr>
<tr>
<td>Encode::Supported</td>
<td>List of Supported Encodings</td>
</tr>
<tr>
<td>Encode::CN</td>
<td>Simplified Chinese Encodings</td>
</tr>
<tr>
<td>Encode::JP</td>
<td>Japanese Encodings</td>
</tr>
<tr>
<td>Encode::KR</td>
<td>Korean Encodings</td>
</tr>
<tr>
<td>Encode::TW</td>
<td>Traditional Chinese Encodings</td>
</tr>
</tbody>
</table>

DESCRIPTION

The Encode module provides the interfaces between Perl's strings and the rest of the system. Perl strings are sequences of characters.

The repertoire of characters that Perl can represent is at least that defined by the Unicode Consortium. On most platforms the ordinal values of the characters (as returned by \( \text{ord}(\text{ch}) \)) is the "Unicode codepoint" for the character (the exceptions are those platforms where the legacy encoding is some variant of EBCDIC rather than a super-set of ASCII - see perlencoded).

Traditionally, computer data has been moved around in 8-bit chunks often called "bytes". These chunks are also known as "octets" in networking standards. Perl is widely used to manipulate data of many types - not only strings of characters representing human or computer languages but also "binary" data being the machine's representation of numbers, pixels in an image - or just about anything.

When Perl is processing "binary data", the programmer wants Perl to process "sequences of bytes". This is not a problem for Perl - as a byte has 256 possible values, it easily fits in Perl's much larger "logical character".

TERMINOLOGY

- **character**: a character in the range 0..(2**32-1) (or more). (What Perl's strings are made of.)
- **byte**: a character in the range 0..255 (A special case of a Perl character.)
- **octet**: 8 bits of data, with ordinal values 0..255 (Term for bytes passed to or from a non-Perl context, e.g. a disk file.)

PERL ENCODING API

\[
$\text{octets} = \text{encode}(\text{ENCODING}, \text{string} [, \text{CHECK}])
\]

Encodes a string from Perl's internal form into ENCODING and returns a sequence of octets. ENCODING can be either a canonical name or an alias. For encoding names and aliases, see Defining Aliases. For CHECK, see Handling Malformed Data.

For example, to convert a string from Perl's internal format to iso-8859-1 (also known as Latin1),
$octets = encode("iso-8859-1", $string);

CAVEAT: When you run $octets = encode("utf8", $string), then $octets may not be equal to $string. Though they both contain the same data, the UTF8 flag for $octets is always off. When you encode anything, UTF8 flag of the result is always off, even when it contains completely valid utf8 string. See The UTF8 flag below.

If the $string is undef then undef is returned.

$string = decode(ENCODING, $octets [, CHECK])

Decodes a sequence of octets assumed to be in ENCODING into Perl's internal form and returns the resulting string. As in encode(), ENCODING can be either a canonical name or an alias. For encoding names and aliases, see Defining Aliases. For CHECK, see Handling Malformed Data.

For example, to convert ISO-8859-1 data to a string in Perl's internal format:

$string = decode("iso-8859-1", $octets);

CAVEAT: When you run $string = decode("utf8", $octets), then $string may not be equal to $octets. Though they both contain the same data, the UTF8 flag for $string is on unless $octets entirely consists of ASCII data (or EBCDIC on EBCDIC machines). See The UTF8 flag below.

If the $string is undef then undef is returned.

[$obj =] find_encoding(ENCODING)

Returns the encoding object corresponding to ENCODING. Returns undef if no matching ENCODING is find.

This object is what actually does the actual (en)decoding.

    $utf8 = decode($name, $bytes);

is in fact

    $utf8 = do{
        $obj = find_encoding($name);
        croak qq(encoding "$name" not found) unless ref $obj;
        $obj->decode($bytes)
    };

with more error checking.

Therefore you can save time by reusing this object as follows;

    my $enc = find_encoding("iso-8859-1");
    while(<>){
        my $utf8 = $enc->decode($_);
        # and do something with $utf8;
    }

Besides ->decode and ->encode, other methods are available as well. For instance, ->name returns the canonical name of the encoding object.

    find_encoding("latin1")->name; # iso-8859-1

See Encode::Encoding for details.

[$length =] from_to($octets, FROM_ENC, TO_ENC [, CHECK])

Converts in-place data between two encodings. The data in $octets must be encoded as octets and not as characters in Perl's internal format. For example, to convert ISO-8859-1 data to Microsoft's CP1250 encoding:
from_to($octets, "iso-8859-1", "cp1250");

and to convert it back:

from_to($octets, "cp1250", "iso-8859-1");

Note that because the conversion happens in place, the data to be converted cannot be a string constant; it must be a scalar variable.

from_to() returns the length of the converted string in octets on success, undef on error.

CAVEAT: The following operations look the same but are not quite so:

from_to($data, "iso-8859-1", "utf8"); #1
$data = decode("iso-8859-1", $data); #2

Both #1 and #2 make $data consist of a completely valid UTF-8 string but only #2 turns UTF8 flag on. #1 is equivalent to

$data = encode("utf8", decode("iso-8859-1", $data));

See The UTF8 flag below.

Also note that

from_to($octets, $from, $to, $check);

is equivalent to

$octets = encode($to, decode($from, $octets), $check);

Yes, it does not respect the $check during decoding. It is deliberately done that way. If you need minute control, decode then encode as follows;

$octets = encode($to, decode($from, $octets, $check_from), $check_to);

$octets = encode_utf8($string);

Equivalent to $octets = encode("utf8", $string); The characters that comprise $string are encoded in Perl's internal format and the result is returned as a sequence of octets. All possible characters have a UTF-8 representation so this function cannot fail.

$string = decode_utf8($octets [, CHECK]);

equivalent to $string = decode("utf8", $octets [, CHECK]). The sequence of octets represented by $octets is decoded from UTF-8 into a sequence of logical characters. Not all sequences of octets form valid UTF-8 encodings, so it is possible for this call to fail. For CHECK, see Handling Malformed Data.

Listing available encodings

use Encode;
@list = Encode->encodings();

Returns a list of the canonical names of the available encodings that are loaded. To get a list of all available encodings including the ones that are not loaded yet, say

@all_encodings = Encode->encodings(":all");

Or you can give the name of a specific module.

@with_jp = Encode->encodings("Encode::JP");

When "::" is not in the name, "Encode::" is assumed.
@ebcdic = Encode->encodings("EBCDIC");

To find out in detail which encodings are supported by this package, see Encode::Supported.

### Defining Aliases

To add a new alias to a given encoding, use:

```perl
use Encode;
use Encode::Alias;
define_alias(newName => ENCODING);
```

After that, newName can be used as an alias for ENCODING. ENCODING may be either the name of an encoding or an encoding object.

But before you do so, make sure the alias is nonexistent with resolve_alias(), which returns the canonical name thereof. i.e.

```perl
Encode::resolve_alias("latin1") eq "iso-8859-1" # true
Encode::resolve_alias("iso-8859-12") # false; nonexistent
Encode::resolve_alias($name) eq $name # true if $name is canonical
```

resolve_alias() does not need use Encode::Alias; it can be exported via use Encode qw(resolve_alias).

See Encode::Alias for details.

### Finding IANA Character Set Registry names

The canonical name of a given encoding does not necessarily agree with IANA IANA Character Set Registry, commonly seen as Content-Type: text/plain; charset=whatever. For most cases canonical names work but sometimes it does not (notably 'utf-8-strict').

Therefore as of Encode version 2.21, a new method mime_name() is added.

```perl
use Encode;
my $enc = find_encoding('UTF-8');
warn $enc->name;    # utf-8-strict
warn $enc->mime_name; # UTF-8
```

See also: Encode::Encoding

### Encoding via PerlIO

If your perl supports PerlIO (which is the default), you can use a PerlIO layer to decode and encode directly via a filehandle. The following two examples are totally identical in their functionality.

```perl
# via PerlIO
open my $in, "<:encoding(shiftjis)", $infile or die;
open my $out, ">
:encoding(euc-jp)", $outfile or die;
while(<$in>) { print $out $_; }

# via from_to
open my $in, "<", $infile or die;
open my $out, ">", $outfile or die;
while(<$in>) {
    from_to($_, "shiftjis", "euc-jp", 1);
    print $out $_;
}
```
Unfortunately, it may be that encodings are PerlIO-savvy. You can check if your encoding is supported by PerlIO by calling the perlio_ok method.

```perl
Encode::perlio_ok("hz"); # False
find_encoding("euc-cn")->perlio_ok; # True where PerlIO is available
```

```perl
use Encode qw(perlio_ok);
perlio_ok("euc-jp")
```

Fortunately, all encodings that come with Encode core are PerlIO-savvy except for hz and ISO-2022-kr. For gory details, see `Encode::Encoding` and `Encode::PerlIO`.

### Handling Malformed Data

The optional `CHECK` argument tells Encode what to do when it encounters malformed data. Without `CHECK`, `Encode::FB_DEFAULT (== 0)` is assumed.

As of version 2.12 Encode supports coderef values for `CHECK`. See below.

**NOTE:** Not all encoding support this feature

Some encodings ignore `CHECK` argument. For example, `Encode::Unicode` ignores `CHECK` and it always croaks on error.

Now here is the list of `CHECK` values available:

- `CHECK = Encode::FB_DEFAULT (== 0)`
  - If `CHECK` is 0, (en|de)code will put a `substitution character` in place of a malformed character. When you encode, `<subchar>` will be used. When you decode the code point `0xFFFD` is used. If the data is supposed to be UTF-8, an optional lexical warning (category `utf8`) is given.

- `CHECK = Encode::FB_CROAK (== 1)`
  - If `CHECK` is 1, methods will die on error immediately with an error message. Therefore, when `CHECK` is set to 1, you should trap the error with `eval()` unless you really want to let it die.

- `CHECK = Encode::FB_QUIET`
  - If `CHECK` is set to `Encode::FB_QUIET`, (en|de)code will immediately return the portion of the data that has been processed so far when an error occurs. The data argument will be overwritten with everything after that point (that is, the unprocessed part of data). This is handy when you have to call decode repeatedly in the case where your source data may contain partial multi-byte character sequences, (i.e. you are reading with a fixed-width buffer). Here is a sample code that does exactly this:

```perl
my $buffer = ''; my $string = '';
while(read $fh, $buffer, 256, length($buffer)){
    $string .= decode($encoding, $buffer, Encode::FB_QUIET);
    # $buffer now contains the unprocessed partial character
}
```

- `CHECK = Encode::FB_WARN`
  - This is the same as above, except that it warns on error. Handy when you are debugging the mode above.

- `perlqq` mode (`CHECK = Encode::FB_PERLQQ`)
- `HTML charref` mode (`CHECK = Encode::FB_HTMLCREF`)
- `XML charref` mode (`CHECK = Encode::FB_XMLCREF`)

For encodings that are implemented by `Encode::XS`, `CHECK == Encode::FB_PERLQQ` turns (en|de)code into `perlqq` fallback mode.
When you decode, \xHHH will be inserted for a malformed character, where HH is the hex representation of the octet that could not be decoded to utf8. And when you encode, \x{HHHH} will be inserted, where HHHH is the Unicode ID of the character that cannot be found in the character repertoire of the encoding.

HTML/XML character reference modes are about the same, in place of \x{HHHH}, HTML uses &#NNN; where NNN is a decimal number and XML uses &#xHHHH; where HHHH is the hexadecimal number.

In Encode 2.10 or later, LEAVE_SRC is also implied.

The bitmask

These modes are actually set via a bitmask. Here is how the FB_XX constants are laid out. You can import the FB_XX constants via use Encode qw(:fallbacks); you can import the generic bitmask constants via use Encode qw(:fallback_all).

FB_DEFAULT FB_CROAK FB_QUIET FB_WARN FB_PERLQQ
DIE_ON_ERR 0x0001 X
WARN_ON_ERR 0x0002 X
RETURN_ON_ERR 0x0004 X X
LEAVE_SRC 0x0008 X
PERLQQ 0x0100 X
HTMLCREF 0x0200
XMLCREF 0x0400

Encode::LEAVE_SRC

If the Encode::LEAVE_SRC bit is not set, but CHECK is, then the second argument to encode() or decode() may be assigned to by the functions. If you’re not interested in this, then bitwise-or the bitmask with it.

As of Encode 2.12 CHECK can also be a code reference which takes the ord value of unmapped caharacter as an argument and returns a string that represents the fallback character. For instance,

```perl
$ascii = encode("ascii", $utf8, sub{ sprintf "<U+%04X>", shift });
```

Acts like FB_PERLQQ but &lt;U+XXXX&gt; is used instead of \x{XXXX}.

**Defining Encodings**

To define a new encoding, use:

```perl
use Encode qw(define_encoding);
define_encoding($object, 'canonicalName' [, alias...]);
```

canonicalName will be associated with $object. The object should provide the interface described in Encode::Encoding. If more than two arguments are provided then additional arguments are taken as aliases for $object.

See Encode::Encoding for more details.

**The UTF8 flag**

Before the introduction of Unicode support in perl, The eq operator just compared the strings represented by two scalars. Beginning with perl 5.8, eq compares two strings with simultaneous consideration of the UTF8 flag. To explain why we made it so, I will quote page 402 of Programming Perl, 3rd ed.

Goal #1:

Old byte-oriented programs should not spontaneously break on the old byte-oriented data they used to work on.
Goal #2:
Old byte-oriented programs should magically start working on the new character-oriented data when appropriate.

Goal #3:
Programs should run just as fast in the new character-oriented mode as in the old byte-oriented mode.

Goal #4:
Perl should remain one language, rather than forking into a byte-oriented Perl and a character-oriented Perl.

Back when Programming Perl, 3rd ed. was written, not even Perl 5.6.0 was born and many features documented in the book remained unimplemented for a long time. Perl 5.8 corrected this and the introduction of the UTF8 flag is one of them. You can think of this perl notion as of a byte-oriented mode (UTF8 flag off) and a character-oriented mode (UTF8 flag on).

Here is how Encode takes care of the UTF8 flag.

- When you encode, the resulting UTF8 flag is always off.
- When you decode, the resulting UTF8 flag is on unless you can unambiguously represent data. Here is the definition of dis-ambiguity.

After $utf8 = decode('foo', $octet);
When $octet is...  The UTF8 flag in $utf8 is
-------------------------------
In ASCII only (or EBCDIC only) OFF
In ISO-8859-1 ON
In any other Encoding ON
-------------------------------

As you see, there is one exception, In ASCII. That way you can assume Goal #1. And with Encode Goal #2 is assumed but you still have to be careful in such cases mentioned in CAVEAT paragraphs.

This UTF8 flag is not visible in perl scripts, exactly for the same reason you cannot (or you don't have to) see if a scalar contains a string, integer, or floating point number. But you can still peek and poke these if you will. See the section below.

**Messing with Perl's Internals**

The following API uses parts of Perl's internals in the current implementation. As such, they are efficient but may change.

- **is_utf8**(STRING [, CHECK])
  - [INTERNAL] Tests whether the UTF8 flag is turned on in the STRING. If CHECK is true, also checks the data in STRING for being well-formed UTF-8. Returns true if successful, false otherwise.
  - As of perl 5.8.1, utf8 also has utf8::is_utf8().

- **_utf8_on**(STRING)
  - [INTERNAL] Turns on the UTF8 flag in STRING. The data in STRING is not checked for being well-formed UTF-8. Do not use unless you know that the STRING is well-formed UTF-8. Returns the previous state of the UTF8 flag (so please don't treat the return value as indicating success or failure), or undef if STRING is not a string.

- **_utf8_off**(STRING)
  - [INTERNAL] Turns off the UTF8 flag in STRING. Do not use frivolously. Returns the previous
state of the UTF8 flag (so please don't treat the return value as indicating success or failure), or
undef if STRING is not a string.

UTF-8 vs. utf8 vs. UTF8

...We now view strings not as sequences of bytes, but as sequences
of numbers in the range 0 .. 2**32-1 (or in the case of 64-bit
computers, 0 .. 2**64-1) -- Programming Perl, 3rd ed.

That has been the perl's notion of UTF-8 but official UTF-8 is more strict; Its ranges is much narrower
(0 .. 10FFFF), some sequences are not allowed (i.e. Those used in the surrogate pair, 0xFFFE, et al).

Now that is overruled by Larry Wall himself.

From: Larry Wall <larry@wall.org>
Date: December 04, 2004 11:51:58 JST
To: perl-unicode@perl.org
Subject: Re: Make Encode.pm support the real UTF-8
Message-Id: <20041204025158.GA28754@wall.org>

On Fri, Dec 03, 2004 at 10:12:12PM +0000, Tim Bunce wrote:
: I've no problem with 'utf8' being perl's unrestricted utf8 encoding,
: but "UTF-8" is the name of the standard and should give the
: corresponding behaviour.

For what it's worth, that's how I've always kept them straight in my
head.

Also for what it's worth, Perl 6 will mostly default to strict but
make it easy to switch back to lax.

Larry

Do you copy? As of Perl 5.8.7, UTF-8 means strict, official UTF-8 while utf8 means liberal, lax,
version thereof. And Encode version 2.10 or later thus groks the difference between UTF-8 and
C"utf8".

    encode("utf8", "\x{FFFF_FFFF}", 1); # okay
    encode("UTF-8", "\x{FFFF_FFFF}", 1); # croaks

UTF-8 in Encode is actually a canonical name for utf-8-strict. Yes, the hyphen between "UTF" and
"8" is important. Without it Encode goes "liberal"

    find_encoding("UTF-8")->name # is 'utf-8-strict'
    find_encoding("utf-8")->name # ditto. names are case insensitive
    find_encoding("utf_8")->name # ditto. "_" are treated as "-
    find_encoding("UTF8")->name # is 'utf8'.

The UTF8 flag is internally called UTF8, without a hyphen. It indicates whether a string is internally
encoded as utf8, also without a hyphen.

SEE ALSO

    Encode::Encoding, Encode::Supported, Encode::PerlIO, encoding, perlebcdic, "open" in perlfunc,
    perlunicode, perluniintro, perlunifaq, perlunitut utf8, the Perl Unicode Mailing List <
    perl-unicode@perl.org>
This project was originated by Nick Ing-Simmons and later maintained by Dan Kogai <dankogai@dan.co.jp>. See AUTHORS for a full list of people involved. For any questions, use <perl-unicode@perl.org> so we can all share.

While Dan Kogai retains the copyright as a maintainer, the credit should go to all those involved. See AUTHORS for those submitted codes.

COPYRIGHT
Copyright 2002-2006 Dan Kogai <dankogai@dan.co.jp>
This library is free software; you can redistribute it and/or modify it under the same terms as Perl itself.