NAME

Encode::Supported -- Encodings supported by Encode

DESCRIPTION

Encoding Names

Encoding names are case insensitive. White space in names is ignored. In addition, an encoding may have aliases. Each encoding has one ''canonical'' name. The ''canonical'' name is chosen from the names of the encoding by picking the first in the following sequence (with a few exceptions).

- The name used by the Perl community. That includes 'utf8' and 'ascii'. Unlike aliases, canonical names directly reach the method so such frequently used words like 'utf8' don't need to do alias lookups.
- The MIME name as defined in IETF RFCs. This includes all ''iso-''s.
- The name in the IANA registry.
- The name used by the organization that defined it.

In case *de jure* canonical names differ from that of the Encode module, they are always aliased if it ever be implemented. So you can safely tell if a given encoding is implemented or not just by passing the canonical name.

Because of all the alias issues, and because in the general case encodings have state, ''Encode'' uses an encoding object internally once an operation is in progress.

Supported Encodings

As of Perl 5.8.0, at least the following encodings are recognized. Note that unless otherwise specified, they are all case insensitive (via alias) and all occurrence of spaces are replaced with '-'. In other words, ''ISO 8859 1'' and ''iso-8859-1'' are identical.

Encodings are categorized and implemented in several different modules but you don't have to `use Encode::XX`to mak e them available for most cases. Encode.pm will automatically load those modules on demand.

Built-in Encodings

The following encodings are always available.

```
Canonical Aliases Comments & References
-----------------------------------------------------------------
ascii US-ascii ISO-646-US [ECMA]
ascii-ctrl Special Encoding
iso-8859-1 latin1 [ISO]
null Special Encoding
utf8 UTF-8 [RFC2279]
-----------------------------------------------------------------
```

null and *ascii-ctrl* are special. ''null'' fails for all character so when you set fallback mode to PERLQQ, HTMLCREF or XMLCREF, ALL CHARACTERS will fall back to character references. Ditto for ''ascii-ctrl'' except for control characters. For fallback modes, see Encode.

Encode::Unicode — other Unicode encodings

Unicode coding schemes other than native utf8 are supported by Encode::Unicode, which will be autoloaded on demand.

```
-----------------------------------------------------------------
UCS-2BE UCS-2, iso-10646-1 [IANA, UC]
UCS-2LE [UC]
UTF-16 [UC]
UTF-16BE [UC]
UTF-16LE [UC]
UTF-32 [UC]
UTF-32BE UCS-4 [UC]
UTF-32LE [UC]
UTF-7 [RFC2152]
-----------------------------------------------------------------
```

To find how (UCS-2|UTF-(16|32))(LE|BE)? differ from one another, see Encode::Unicode.

UTF-7 is a special encoding which ''re-encodes'' UTF-16BE into a 7-bit encoding. It is implemented separately by Encode::Unicode::UTF7.

Encode::Byte — Extended ASCII

Encode::Byte implements most single-byte encodings except for Symbols and EBCDIC. The following encodings are based on single-byte encodings implemented as extended ASCII. Most of them map x80-xff (upper half) to non-ASCII characters.

ISO-8859 and corresponding vendor mappings

Since there are so many, they are presented in table format with languages and corresponding encoding names by vendors. Note that the table is sorted in order of ISO-8859 and the corresponding vendor mappings are slightly different from that of ISO. See <http://czyborra.com/charsets/iso8859.html> for details.

```
Lang/Regions ISO/Other Std. DOS Windows Macintosh Others
-----------------------------------------------------------------
N. America (ASCII) cp437 AdobeStandardEncoding
cp863 (DOSCanadaF)
W. Europe iso-8859-1 cp850 cp1252 MacRoman nextstep
hp-roman8
cp860 (DOSPortuguese)
Cntrl. Europe iso-8859-2 cp852 cp1250 MacCentralEurRoman
MacCroatian
MacRomanian
MacRumanian
Latin3[1] iso-8859-3
Latin4[2] iso-8859-4
Cyrillics iso-8859-5 cp855 cp1251 MacCyrillic
(See also next section) cp866 MacUkrainian
Arabic iso-8859-6 cp864 cp1256 MacArabic
cp1006 MacFarsi
Greek iso-8859-7 cp737 cp1253 MacGreek
cp869 (DOSGreek2)
Hebrew iso-8859-8 cp862 cp1255 MacHebrew
Turkish iso-8859-9 cp857 cp1254 MacTurkish
Nordics iso-8859-10 cp865
cp861 MacIcelandic
MacSami
Thai iso-8859-11[3] cp874 MacThai
(iso-8859-12 is nonexistent. Reserved for Indics?)
Baltics iso-8859-13 cp775 cp1257
Celtics iso-8859-14
Latin9 [4] iso-8859-15
```

```
Latin10 iso-8859-16
Vietnamese viscii cp1258 MacVietnamese
------------------------------------------------------------------

[1] Esperanto, Maltese, and Turkish. Turkish is now on 8859-9.
[2] Baltics. Now on 8859-10, except for Latvian.
[3] TIS 620 + Non-Breaking Space (0xA0 / U+00A0)
[4] Nicknamed Latin0; the Euro sign as well as French and Finnish
letters that are missing from 8859-1 were added.
```

All cp* are also available as ibm-*, ms-*, and windows-* . See also <http://czyborra.com/charsets/codepages.html>.

Macintosh encodings don't seem to be registered in such entities as IANA. ''Canonical'' names in Encode are based upon Apple's Tech Note 1150. See <http://developer.apple.com/technotes/tn/tn1150.html> for details.

KOI8 - De Facto Standard for the Cyrillic world

Though ISO-8859 does have ISO-8859-5, the KOI8 series is far more popular in the Net. Encode comes with the following KOI charsets. For gory details, see <http://czyborra.com/charsets/cyrillic.html>

```
------------------------------------------------------------------
koi8-f
koi8-r cp878 [RFC1489]
koi8-u [RFC2319]
------------------------------------------------------------------
```

gsm0338 – Hentai Latin 1

GSM0338 is for GSM handsets. Though it shares alphanumerals with ASCII, control character ranges and other parts are mapped very differently, mainly to store Greek characters. There are also escape sequences (starting with 0x1B) to cover e.g. the Euro sign.

This was once handled by Encode::Bytes but because of all those unusual specifications, Encode 2.20 has relocated the support to Encode::GSM0338. See Encode::GSM0338 for details.

gsm0338 support before 2.19

Some special cases like a trailing 0x00 byte or a lone 0x1B byte are not well-defined and *decode()* will return an empty string for them. One possible workaround is

```
$gsm = s/\x00\z/\x00\x00/;
$uni = decode("gsm0338", $gsm);
$uni .= "\xA0" if $gsm = /\x1B\z/;
```

Note that the Encode implementation of GSM0338 does not implement the reuse of Latin capital letters as Greek capital letters (for example, the 0x5A is U+005A (LATIN CAPITAL LETTER Z), not U+0396 (GREEK CAPITAL LETTER ZETA).

The GSM0338 is also covered in Encode::Byte even though it is not an ''extended ASCII'' encoding.

CJK: Chinese, Japanese, Korean (Multibyte)

Note that Vietnamese is listed above. Also read ''Encoding vs Charset'' below. Also note that these are implemented in distinct modules by countries, due to the size concerns (simplified Chinese is mapped to 'CN', continental China, while traditional Chinese is mapped to 'TW', Taiwan). Please refer to their respective documentation pages.

Encode::CN — Continental China

```
Standard DOS/Win Macintosh Comment/Reference
-------------------------------------------------------------------
euc-cn [1] MacChineseSimp
(gbk) cp936 [2]
gb12345-raw { GB12345 without CES }
gb2312-raw { GB2312 without CES }
hz
iso-ir-165
-------------------------------------------------------------------

[1] GB2312 is aliased to this. See L<Microsoft-related naming mess>
[2] gbk is aliased to this. See L<Microsoft-related naming mess>
```

Encode::JP — Japan

```
Standard DOS/Win Macintosh Comment/Reference
-------------------------------------------------------------------
euc-jp
shiftjis cp932 macJapanese
7bit-jis
iso-2022-jp [RFC1468]
iso-2022-jp-1 [RFC2237]
jis0201-raw { JIS X 0201 (roman + halfwidth kana) without CES }
jis0208-raw { JIS X 0208 (Kanji + fullwidth kana) without CES }
jis0212-raw { JIS X 0212 (Extended Kanji) without CES }
-------------------------------------------------------------------
```

Encode::KR — Korea

```
Standard DOS/Win Macintosh Comment/Reference
-------------------------------------------------------------------
euc-kr MacKorean [RFC1557]
cp949 [1]
iso-2022-kr [RFC1557]
johab [KS X 1001:1998, Annex 3]
ksc5601-raw { KSC5601 without CES }
-------------------------------------------------------------------

[1] ks_c_5601-1987, (x-)?windows-949, and uhc are aliased to this.
See below.
```

Encode::TW — Taiwan

```
Standard DOS/Win Macintosh Comment/Reference
-------------------------------------------------------------------
big5-eten cp950 MacChineseTrad {big5 aliased to big5-eten}
big5-hkscs
-------------------------------------------------------------------
```

Encode::HanExtra — More Chinese via CPAN

Due to the size concerns, additional Chinese encodings below are distributed separately on CPAN, under the name Encode::HanExtra.

```
Standard DOS/Win Macintosh Comment/Reference
-----------------------------------------------------------------
big5ext CMEX's Big5e Extension
big5plus CMEX's Big5+ Extension
cccii Chinese Character Code for Information Interchange
euc-tw EUC (Extended Unix Character)
gb18030 GBK with Traditional Characters
-----------------------------------------------------------------
```

Encode::JIS2K — JIS X 0213 encodings via CPAN

Due to size concerns, additional Japanese encodings below are distributed separately on CPAN, under the name Encode::JIS2K.

```
Standard DOS/Win Macintosh Comment/Reference
-----------------------------------------------------------------
euc-jisx0213
shiftjisx0123
iso-2022-jp-3
jis0213-1-raw
jis0213-2-raw
-----------------------------------------------------------------
```

Miscellaneous encodings

Encode::EBCDIC

See perlebcdic(1) for details.

```
-----------------------------------------------------------------
cp37
cp500
cp875
cp1026
cp1047
posix-bc
-----------------------------------------------------------------
```

Encode::Symbols

For symbols and dingbats.

```
-----------------------------------------------------------------
symbol
dingbats
MacDingbats
AdobeZdingbat
AdobeSymbol
-----------------------------------------------------------------
```

Encode::MIME::Header

Strictly speaking, MIME header encoding documented in RFC 2047 is more of encapsulation than encoding. However, their support in modern world is imperative so they are supported.

```
-----------------------------------------------------------------
MIME-Header [RFC2047]
MIME-B [RFC2047]
MIME-Q [RFC2047]
-----------------------------------------------------------------
```

Encode::Guess

This one is not a name of encoding but a utility that lets you pick up the most appropriate encoding for a data out of given *suspects*. See Encode::Guess for details.

Unsupported encodings

The following encodings are not supported as yet; some because they are rarely used, some because of technical difficulties. They may be supported by external modules via CPAN in the future, however.

ISO-2022-JP-2 [RFC1554]
: Not very popular yet. Needs Unicode Database or equivalent to implement *encode()* (because it includes JIS X 0208/0212, KSC5601, and GB2312 simultaneously, whose code points in Unicode overlap. So you need to lookup the database to determine to what character set a given Unicode character should belong).

ISO-2022-CN [RFC1922]
: Not very popular. Needs CNS 11643-1 and -2 which are not available in this module. CNS 11643 is supported (via euc-tw) in Encode::HanExtra. Autrijus Tang may add support for this encoding in his module in future.

Various HP-UX encodings
: The following are unsupported due to the lack of mapping data.

```
'8' - arabic8, greek8, hebrew8, kana8, thai8, and turkish8
'15' - japanese15, korean15, and roi15
```

Cyrillic encoding ISO-IR-111
: Anton Tagunov doubts its usefulness.

ISO-8859-8-1 [Hebrew]
: None of the Encode team knows Hebrew enough (ISO-8859-8, cp1255 and MacHebrew are supported because and just because there were mappings available at <http://www.unicode.org/>).
Contributions welcome.

ISIRI 3342, Iran System, ISIRI 2900 [Farsi]
: Ditto.

Thai encoding TCVN
: Ditto.

Vietnamese encodings VPS
: Though Jungshik Shin has reported that Mozilla supports this encoding, it was too late before 5.8.0 for us to add it. In the future, it may be available via a separate module. See <http://lxr.mozilla.org/seamonkey/source/intl/uconv/ucvlatin/vps.uf> and <http://lxr.mozilla.org/seamonkey/source/intl/uconv/ucvlatin/vps.ut> if you are interested in helping us.

Various Mac encodings
: The following are unsupported due to the lack of mapping data.

```
MacArmenian, MacBengali, MacBurmese, MacEthiopic
MacExtArabic, MacGeorgian, MacKannada, MacKhmer
MacLaotian, MacMalayalam, MacMongolian, MacOriya
MacSinhalese, MacTamil, MacTelugu, MacTibetan
MacVietnamese
```

The rest which are already available are based upon the vendor mappings at <http://www.unicode.org/Public/MAPPINGS/VENDORS/APPLE/>

(Mac) Indic encodings
: The maps for the following are available at <http://www.unicode.org/> but remain unsupported because those encodings need an algorithmical approach, currently unsupported by *enc2xs*:

```
MacDevanagari
MacGurmukhi
MacGujarati
```

For details, please seeUnicode mapping issues and notes: at <http://www.unicode.org/Public/MAPPINGS/VENDORS/APPLE/DEVANAGA.TXT>

I believe this issue is prevalent not only for Mac Indics but also in other Indic encodings, but the above were the only Indic encodings maps that I could find at <http://www.unicode.org/>

Encoding vs. Charset — terminology

We are used to using the term (character) *encoding* and *character set*in terchangeably. But just as confusing the terms byte and character is dangerous and the terms should be differentiated when needed, we need to differentiate *encoding* and *character set*.

To understand that, here is a description of how we make computers grok our characters.

- First we start with which characters to include. We call this collection of characters *character repertoire*.
- Then we have to give each character a unique ID so your computer can tell the difference between 'a' and 'A'. This itemized character repertoire is now a *character set*.
- If your computer can grow the character set without further processing, you can go ahead and use it. This is called a *coded character set*(CCS) or *r aw character encoding*. ASCII is used this way for most cases.
- But in many cases, especially multi-byte CJK encodings, you have to tweak a little more. Your network connection may not accept any data with the Most Significant Bit set, and your computer may not be able to tell if a given byte is a whole character or just half of it. So you have to *encode* the character set to use it.

  A *character encoding scheme*(CES) determines ho w to encode a given character set, or a set of multiple character sets. 7bit ISO-2022 is an example of a CES. You switch between character sets via *escape sequences*.

Technically, or mathematically, speaking, a character set encoded in such a CES that maps character by character may form a CCS. EUC is such an example. The CES of EUC is as follows:

- Map ASCII unchanged.
- Map such a character set that consists of 94 or 96 powered by N members by adding 0x80 to each byte.
- You can also use 0x8e and 0x8f to indicate that the following sequence of characters belongs to yet another character set. To each following byte is added the value 0x80.

By carefully looking at the encoded byte sequence, you can find that the byte sequence conforms a unique number. In that sense, EUC is a CCS generated by a CES above from up to four CCS (complicated?). UTF-8 falls into this category. See "UTF-8" in perlUnicode to find out how UTF-8 maps Unicode to a byte sequence.

You may also have found out by now why 7bit ISO-2022 cannot comprise a CCS. If you look at a byte sequence x21x21, you can't tell if it is two !'s or IDEOGRAPHIC SPACE. EUC maps the latter to xA1xA1 so you have no trouble differentiating between "!!". and " ".

Encoding Classification (by Anton Tagunov and Dan Kogai)

This section tries to classify the supported encodings by their applicability for information exchange over the Internet and to choose the most suitable aliases to name them in the context of such communication.

- To (en|de)code encodings marked by `(**)`, you need `Encode::HanExtra` available from CPAN.

Encoding names

```
US-ASCII UTF-8 ISO-8859-* KOI8-R
Shift_JIS EUC-JP ISO-2022-JP ISO-2022-JP-1
EUC-KR Big5 GB2312
```

are registered with IANA as preferred MIME names and may be used over the Internet.

`Shift_JIS` has been officialized by JIS X 0208:1997. ''Microsoft-related naming mess'' gives details.

`GB2312` is the IANA name for `EUC-CN`. See ''Microsoft-related naming mess'' for details.

`GB_2312-80` *raw* encoding is available as `gb2312-raw` with Encode. See Encode::CN for details.

```
EUC-CN
KOI8-U [RFC2319]
```

have not been registered with IANA (as of March 2002) but seem to be supported by major web browsers. The IANA name for `EUC-CN` is `GB2312`.

```
KS_C_5601-1987
```

is heavily misused. See ''Microsoft-related naming mess'' for details.

`KS_C_5601-1987` *raw* encoding is available as `kcs5601-raw` with Encode. See Encode::KR for details.

```
UTF-16 UTF-16BE UTF-16LE
```

are IANA-registered `charsets`. See [RFC 2781] for details. Jungshik Shin reports that UTF-16 with a BOM is well accepted by MS IE 5/6 and NS 4/6. Beware however that

- `UTF-16` support in any software you're going to be using/interoperating with has probably been less tested then `UTF-8` support
- `UTF-8` coded data seamlessly passes traditional command piping (`cat`, `more`, etc.) while `UTF-16` coded data is likely to cause confusion (with its zero bytes, for example)
- it is beyond the power of words to describe the way HTML browsers encode non-`ASCII` form data. To get a general impression, visit <http://www.alanflavell.org.uk/charset/form-i18n.html>. While encoding of form data has stabilized for `UTF-8` encoded pages (at least IE 5/6, NS 6, and Opera 6 behave consistently), be sure to expect fun (and cross-browser discrepancies) with `UTF-16` encoded pages!

The rule of thumb is to use `UTF-8` unless you know what you're doing and unless you really benefit from using `UTF-16`.

```
ISO-IR-165 [RFC1345]
VISCII
GB 12345
GB 18030 (**) (see links bellow)
EUC-TW (**)
```

are totally valid encodings but not registered at IANA. The names under which they are listed here are probably the most widely-known names for these encodings and are recommended names.

```
BIG5PLUS (**)
```

is a proprietary name.

Microsoft-related naming mess

Microsoft products misuse the following names:

KS_C_5601-1987
: Microsoft extension to `EUC-KR`.

  Proper names: `CP949`, `UHC`, `x-windows-949` (as used by Mozilla).

  See <http://lists.w3.org/Archives/Public/ietf-charsets/2001AprJun/0033.html> for details.

  Encode aliases `KS_C_5601-1987` to `cp949` to reflect this common misusage. *Raw*

`KS_C_5601-1987` encoding is available as `kcs5601-raw`.

See Encode::KR for details.

GB2312
: Microsoft extension to `EUC-CN`.

  Proper names: `CP936`, `GBK`.

  `GB2312` has been registered in the `EUC-CN` meaning at IANA. This has partially repaired the situation: Microsoft's `GB2312` has become a superset of the official `GB2312`.

  Encode aliases `GB2312` to `euc-cn` in full agreement with IANA registration. `cp936` is supported separately. *Raw* `GB_2312-80` encoding is available as `gb2312-raw`.

  See Encode::CN for details.

Big5
: Microsoft extension to `Big5`.

  Proper name: `CP950`.

  Encode separately supports `Big5` and `cp950`.

Shift_JIS
: Microsoft's understanding of `Shift_JIS`.

  JIS has not endorsed the full Microsoft standard however. The official`Shift_JIS` includes only JIS X 0201 and JIS X 0208 character sets, while Microsoft has always used `Shift_JIS` to encode a wider character repertoire. See `IANA` registration for `Windows-31J`.

  As a historical predecessor, Microsoft's variant probably has more rights for the name, though it may be objected that Microsoft shouldn't have used JIS as part of the name in the first place.

  Unambiguous name: `CP932`. `IANA` name (also used by Mozilla, and provided as an alias by Encode): `Windows-31J`.

  Encode separately supports `Shift_JIS` and `cp932`.

Glossary

character repertoire
: A collection of unique characters. A *character* set in the strictest sense. At this stage, characters are not numbered.

coded character set (CCS)
: A character set that is mapped in a way computers can use directly. Many character encodings, including EUC, fall in this category.

character encoding scheme (CES)
: An algorithm to map a character set to a byte sequence. You don't have to be able to tell which character set a given byte sequence belongs. 7-bit ISO-2022 is a CES but it cannot be a CCS. EUC is an example of being both a CCS and CES.

charset (in MIME context)
: has long been used in the meaning of `encoding`, CES.

  While the word combination `character set` has lost this meaning in MIME context since [RFC 2130], the `charset` abbreviation has retained it. This is how [RFC 2277] and [RFC 2278] bless `charset`:

```
This document uses the term "charset" to mean a set of rules for
mapping from a sequence of octets to a sequence of characters, such
as the combination of a coded character set and a character encoding
scheme; this is also what is used as an identifier in MIME "charset="
parameters, and registered in the IANA charset registry ... (Note
that this is NOT a term used by other standards bodies, such as ISO).
[RFC 2277]
```

EUC
: Extended Unix Character. See ISO-2022.

ISO-2022
: A CES that was carefully designed to coexist with ASCII. There are a 7 bit version and an 8 bit version.

  The 7 bit version switches character set via escape sequence so it cannot form a CCS. Since this is more difficult to handle in programs than the 8 bit version, the 7 bit version is not very popular except for iso-2022-jp, the *de facto* standard CES for e-mails.

  The 8 bit version can form a CCS. EUC and ISO-8859 are two examples thereof. Pre-5.6 perl could use them as string literals.

UCS
: Short for *Universal Character Set*. When you say just UCS, it means *Unicode*.

UCS-2
: ISO/IEC 10646 encoding form: Universal Character Set coded in two octets.

Unicode
: A character set that aims to include all character repertoires of the world. Many character sets in various national as well as industrial standards have become, in a way, just subsets of Unicode.

UTF
: Short for *Unicode Transformation Format*. Determines how to map a Unicode character into a byte sequence.

UTF-16
: A UTF in 16-bit encoding. Can either be in big endian or little endian. The big endian version is called UTF-16BE (equal to UCS-2 + surrogate support) and the little endian version is called UTF-16LE.

See Also

Encode, Encode::Byte, Encode::CN, Encode::JP, Encode::KR, Encode::TW, Encode::EBCDIC, Encode::Symbol Encode::MIME::Header, Encode::Guess

References

ECMA
: European Computer Manufacturers Association <http://www.ecma.ch>

  ECMA-035 (eq `ISO-2022`)
  : <http://www.ecma.ch/ecma1/STAND/ECMA-035.HTM>

    The specification of ISO-2022 is available from the link above.

IANA
: Internet Assigned Numbers Authority <http://www.iana.org/>

  Assigned Charset Names by IANA
  : <http://www.iana.org/assignments/character-sets>

    Most of the `canonical names` in Encode derive from this list so you can directly apply the string you have extracted from MIME header of mails and web pages.

ISO
International Organization for Standardization <http://www.iso.ch/>

RFC
Request For Comments — need I say more? < http://www.rfc-editor.org/>, <http://www.ietf.org/rfc.html>, <http://www.faqs.org/rfcs/>

UC
Unicode Consortium <http://www.unicode.org/>

Unicode Glossary
<http://www.unicode.org/glossary/>

The glossary of this document is based upon this site.

Other Notable Sites

czyborra.com
<http://czyborra.com/>

Contains a lot of useful information, especially gory details of ISO vs. vendor mappings.

CJK.inf
<http://examples.oreilly.com/cjkvinfo/doc/cjk.inf>

Somewhat obsolete (last update in 1996), but still useful. Also try

<ftp://ftp.oreilly.com/pub/examples/nutshell/cjkv/pdf/GB18030_Summary.pdf>

You will find brief info on `EUC-CN`, `GBK` and mostly on `GB 18030`.

Jungshik Shin's Hangul FAQ
<http://jshin.net/faq>

And especially its subject 8.

<http://jshin.net/faq/qa8.html>

A comprehensive overview of the Korean (`KS *`) standards.

debian.org: ''Introduction to i18n''
A brief description for most of the mentioned CJK encodings is contained in <http://www.debian.org/doc/manuals/intro-i18n/ch-codes.en.html>

Offline sources

`CJKV Information Processing`b y Ken Lunde
CJKV Information Processing 1999 O'Reilly & Associates, ISBN : 1-56592-224-7

The modern successor of `CJK.inf`.

Features a comprehensive coverage of CJKV character sets and encodings along with many other issues faced by anyone trying to better support CJKV languages/scripts in all the areas of information processing.

To purchase this book, visit <http://oreilly.com/catalog/9780596514471/> or your favourite bookstore.