NAME

Locale::Util − Portable l10n and i10n functions

SYNOPSIS

```
use Locale::Util;

my @linguas = parse_http_accept_language $ENV{HTTP_ACCEPT_LANGUAGE};

my @charsets = parse_http_accept_charset $ENV{HTTP_ACCEPT_CHARSET};

# Trie to set the locale to Brasilian Portuguese in UTF−8.
my $set_locale = set_locale LC_ALL, 'pt', 'BR', 'utf−8';

set_locale_cache $last_cache;

my $cache = get_locale_cache;

web_set_locale ($ENV{HTTP_ACCEPT_LANGUAGE}, $ENV_ACCEPT_CHARSET);

web_set_locale (['fr−BE', 'fr', 'it'], ['cp1252', 'utf−8']);
```

DESCRIPTION

This module provides portable functions dealing with localization (l10n) and internationalization(i10n). It doesn't export anything by default, you have to specify each function you need in the import list, or use the fully qualified name.

The functions here have a focus on web development, although they are general enough to have them in the Locale:: namespace.

This module is considered alpha code. The interface is not stable. Please contact the author if you want to use it in production code.

This module was introduced in libintl-perl 1.17.

FUNCTIONS

parse_http_accept_language STRING

Parses a string as passed in the HTTP header ‘‘Accept-Language’’. It returns a list of tokens sorted by the quality value, see RFC 2616 for details.

Example:

```
parse_http_accept ("fr−fr, fr; q=0.7, de; q=0.3");
```

This means: Give me French for France with a quality value of 1.0 (the maximum). Otherwise I will take any other French version (quality 0.7), German has a quality of 0.3 for me.

The function will return a list of tokens in the order of their quality values, in this case ‘‘fr-fr’’, ‘‘fr’’ and ‘‘de’’.

The function is more forgiving than RFC 2616. It accepts quality values greater than 1.0 and with more than 3 decimal places. It also accepts languages and country names with more than 8 characters. The language ‘‘*’’ is translated into ‘‘C’’.

parse_http_accept_charset STRING

Parses a string as passed in the HTTP header ‘‘Accept-Charset’’. It returns a list of tokens sorted by the quality value, see RFC 2616 for details.

The special character set ‘‘*’’ (means all character sets) will be translated to the undefined value.

set_locale CATEGORY, LANGUAGE[, COUNTRY, CHARSET]

Tries to set the user locale by means of **POSIX::setlocale()**. The latter function has the disadvantage, that its second argument (the locale description string) is completely non-standard and system-

dependent. This function tries its best at guessing the system's notion of a locale dientifier, with the arguments supplied:

CATEGORY
: An integer argument for a valid locale category. These are the LC_* constants (LC_ALL, LC_CTIME, LC_COLLATE, ...) defined in both **Locale::Messages** (3pm) and **POSIX** (3pm).

LANGUAGE
: A 2−letter language identifier as per ISO 639. Case doesn't matter, but an unchanged version (ie. not lower-cased) of the language you provided will always be tried to.

COUNTRY
: A 2−letter language identifier as per ISO 639. Case doesn't matter, but an unchanged version (ie. not lower-cased) of the language you provided will always be tried to.

  This parameter is optional. If it is not defined, the function will try to guess an appropriate country, otherwise leave it to the operating system.

CHARSET
: A valid charset name. Valid means valid! The charset ''utf8'' is not valid (it is ''utf−8''). Charset names that are accepted by the guessing algorithms in **Encode** (3pm) are also not necessarily valid.

  If the parameter is undefined, it is ignored. It is always ignored under Windows.

The function tries to approach the desired locale in loops, refining it on every success. It will first try to set the language (for any country), then try to select the correct language, and finally try to select the correct charset.

The return value is false in case of failure, or the return value of the underlying **POSIX::setlocale()** call in case of success.

In array context, the function returns the country name that was passed in the successful call to **POSIX::setlocale()**. If this string is equal to the country name you passed as an argument, you can be reasonably sure that the settings for this country are really used. If it is not equal, the function has taken a guess at the country (it has a list of ''default'' countries for each language). It seems that under Windows, **POSIX::setlocale()** also succeeds, if you pass a country name that is actually not supported. Therefore, the information is not completely reliable.

Please note that this function is intended for server processes (especially web applications) that need to switch in a portable way to a certain locale. It is **not** the recommended way to set the program locale for a regular application. In a regular application you should do the following:

```
use POSIX qw (setlocale LC_ALL);
setlocale LC_ALL, '';
```

The empty string as the second argument means, that the system should switch to the user's default locale.

get_locale_cache
: The function **set_locale()** is potentially expansive, especially when it fails, because it can try a lot of different combinations, and the system may have to load a lot of locale definitions from its internal database.

  In order to speed up things, results are internally cached in a hash, keys are the languages, subkeys countries, subsubkeys the charsets. You can get a reference to this hash with **get_locale_cache()**.

  The function cannot fail.

set_locale_cache HASH
: Sets the internal cache. You can either pass a hash or a hash reference. The function will use this as its cache, discarding its old cache. This allows you to keep the hash persistent.

  The function cannot fail.

web_set_locale (ACCEPT_LANGUAGE, ACCEPT_CHARSET, CATEGORY, AVAILABLE)

Try to change the locale to the settings described by ACCEPT_LANGUAGE and ACCEPT_CHARSET. For each argument you can either pass a string as in the corresponding http header, or a reference to an array of language resp. charset identifiers.

Currently only the first charset passed is used as an argument. You are strongly encouraged to pass a hard-coded value here, so that you have control about your output.

The argument **CATEGORY** specifies the category (one of the LC_* constants as defined in **Locale::Messages** (3pm) or in **POSIX** (3pm)). The category defaults to LC_ALL.

You can pass an optional reference to a list of locales in XPG4 format that are available in your application. This is useful if you know which languages are supported by your application. In fact, only the language part of the values in the list are considered (for example for ‘‘en_US’’, only ‘‘en’’ is used). The country or other parts are ignored.

The function returns the return value of the underlying **set_locale()** call, or false on failure.

The function returns false on failure. On success it returns the return value of the underlying **set_locale()** call. This value can be used directly in subsequent calls to **POSIX::setlocale()**. In array context, it additionally returns the identifiers for the language, the country, and the charset actually used.

BUGS

The function **set_locale()** probably fails to guess the correct locale identifier on a lot of systems. If you have found such a case, please submit it as a bug report.

The bug tracking system for this packags is at http://rt.cpan.org/NoAuth/Bugs.html?libintl−perl

Please note that this module is considered alpha code, and the interface is not stable. Please contact the author, if you want to use it in production code.

AUTHOR

SEE ALSO

POSIX (3pm), **perl** (1)