NAME

charnames - access to Unicode character names and named character sequences; also define character names

SYNOPSIS

```
use charnames ':full';
print "\N{GREEK SMALL LETTER SIGMA} is called sigma.\n";
print "\N{LATIN CAPITAL LETTER E WITH VERTICAL LINE BELOW}",
      " is an officially named sequence of two Unicode characters\n";

use charnames ':short';
print "\N{greek:Sigma} is an upper-case sigma.\n";

use charnames qw(cyrillic greek);
print "\N{sigma} is Greek sigma, and \N{be} is Cyrillic b.\n";

use charnames ":full", ":alias" => {
  e_ACUTE => "LATIN SMALL LETTER E WITH ACUTE",
  mychar => 0xE8000,  # Private use area
};
print "\N{e_ACUTE} is a small letter e with an acute.\n";
print "\\N{mychar} allows me to name private use characters.\n";

use charnames ();
print charnames::viacode(0x1234); # prints "ETHIOPIC SYLLABLE SEE"
printf "%04X", charnames::vianame("GOTHIC LETTER AHSA"); # prints
                                                         # "10330"
print charnames::vianame("LATIN CAPITAL LETTER A"); # prints 65 on
                                                    # ASCII platforms;
                                                    # 193 on EBCDIC
print charnames::string_vianame("LATIN CAPITAL LETTER A"); # prints "A"
```

DESCRIPTION

Pragma `use charnames` is used to gain access to the names of the Unicode characters and named character sequences, and to allow you to define your own character and character sequence names.

All forms of the pragma enable use of the following 3 functions:

- *charnames::string_vianame(name)* for run-time lookup of a either a character name or a named character sequence, returning its string representation
- *charnames::vianame(name)* for run-time lookup of a character name (but not a named character sequence) to get its ordinal value (code point)
- *charnames::viacode(code)* for run-time lookup of a code point to get its Unicode name.

All forms other than `"use charnames ();"` also enable the use of `\N{`*`CHARNAME`*`}` sequences to compile a Unicode character into a string, based on its name.

Note that `\N{U+...}`, where the ... is a hexadecimal number, also inserts a character into a string, but doesn't require the use of this pragma. The character it inserts is the one whose code point (ordinal value) is equal to the number. For example, `"\N{U+263a}"` is the Unicode (white background, black foreground) smiley face; it doesn't require this pragma, whereas the equivalent, `"\N{WHITE SMILING FACE}"` does. Also, `\N{...}` can mean a regex quantifier instead of a character name, when the ... is a number (or comma separated pair of numbers (see *"QUANTIFIERS" in perlreref*), and is not related to this pragma.

The `charnames` pragma supports arguments `:full`, `:short`, script names and customized aliases. If `:full` is present, for expansion of `\N{CHARNAME}`, the string *CHARNAME* is first looked up in the list of standard Unicode character names. If `:short` is present, and *CHARNAME* has the form `SCRIPT:CNAME`, then *CNAME* is looked up as a letter in script *SCRIPT*. If `use charnames` is used with script name arguments, then for `\N{CHARNAME}` the name *CHARNAME* is looked up as a letter in the given scripts (in the specified order). Customized aliases can override these, and are explained in *CUSTOM ALIASES*.

For lookup of *CHARNAME* inside a given script *SCRIPTNAME* this pragma looks for the names

```
SCRIPTNAME CAPITAL LETTER CHARNAME
SCRIPTNAME SMALL LETTER CHARNAME
SCRIPTNAME LETTER CHARNAME
```

in the table of standard Unicode names. If *CHARNAME* is lowercase, then the `CAPITAL` variant is ignored, otherwise the `SMALL` variant is ignored.

Note that `\N{...}` is compile-time; it's a special form of string constant used inside double-quotish strings; this means that you cannot use variables inside the `\N{...}`. If you want similar run-time functionality, use *charnames::string_vianame()*.

For the C0 and C1 control characters (U+0000..U+001F, U+0080..U+009F) there are no official Unicode names but you can use instead the ISO 6429 names (LINE FEED, ESCAPE, and so forth, and their abbreviations, LF, ESC, ...). In Unicode 3.2 (as of Perl 5.8) some naming changes took place, and ISO 6429 was updated, see *ALIASES*.

If the input name is unknown, `\N{NAME}` raises a warning and substitutes the Unicode REPLACEMENT CHARACTER (U+FFFD).

For `\N{NAME}`, it is a fatal error if `use bytes` is in effect and the input name is that of a character that won't fit into a byte (i.e., whose ordinal is above 255).

Otherwise, any string that includes a `\N{charname}` or `\N{U+code point}` will automatically have Unicode semantics (see *"Byte and Character Semantics" in perlunicode*).

ALIASES

A few aliases have been defined for convenience: instead of having to use the official names

```
LINE FEED (LF)
FORM FEED (FF)
CARRIAGE RETURN (CR)
NEXT LINE (NEL)
```

(yes, with parentheses), one can use

```
LINE FEED
FORM FEED
CARRIAGE RETURN
NEXT LINE
LF
FF
CR
NEL
```

All the other standard abbreviations for the controls, such as `ACK` for `ACKNOWLEDGE` also can be used.

One can also use

```
    BYTE ORDER MARK
    BOM
```

and these abbreviations

```
    Abbreviation        Full Name

    CGJ                 COMBINING GRAPHEME JOINER
    FVS1                MONGOLIAN FREE VARIATION SELECTOR ONE
    FVS2                MONGOLIAN FREE VARIATION SELECTOR TWO
    FVS3                MONGOLIAN FREE VARIATION SELECTOR THREE
    LRE                 LEFT-TO-RIGHT EMBEDDING
    LRM                 LEFT-TO-RIGHT MARK
    LRO                 LEFT-TO-RIGHT OVERRIDE
    MMSP                MEDIUM MATHEMATICAL SPACE
    MVS                 MONGOLIAN VOWEL SEPARATOR
    NBSP                NO-BREAK SPACE
    NNBSP               NARROW NO-BREAK SPACE
    PDF                 POP DIRECTIONAL FORMATTING
    RLE                 RIGHT-TO-LEFT EMBEDDING
    RLM                 RIGHT-TO-LEFT MARK
    RLO                 RIGHT-TO-LEFT OVERRIDE
    SHY                 SOFT HYPHEN
    VS1                 VARIATION SELECTOR-1
    .
    .
    .
    VS256               VARIATION SELECTOR-256
    WJ                  WORD JOINER
    ZWJ                 ZERO WIDTH JOINER
    ZWNJ                ZERO WIDTH NON-JOINER
    ZWSP                ZERO WIDTH SPACE
```

For backward compatibility one can use the old names for certain C0 and C1 controls

```
    old                          new

    FILE SEPARATOR               INFORMATION SEPARATOR FOUR
    GROUP SEPARATOR              INFORMATION SEPARATOR THREE
    HORIZONTAL TABULATION        CHARACTER TABULATION
    HORIZONTAL TABULATION SET    CHARACTER TABULATION SET
    HORIZONTAL TABULATION WITH JUSTIFICATION    CHARACTER TABULATION
                                                WITH JUSTIFICATION
    PARTIAL LINE DOWN            PARTIAL LINE FORWARD
    PARTIAL LINE UP              PARTIAL LINE BACKWARD
    RECORD SEPARATOR             INFORMATION SEPARATOR TWO
    REVERSE INDEX                REVERSE LINE FEED
    UNIT SEPARATOR               INFORMATION SEPARATOR ONE
    VERTICAL TABULATION          LINE TABULATION
    VERTICAL TABULATION SET      LINE TABULATION SET
```

but the old names in addition to giving the character will also give a warning about being deprecated.

And finally, certain published variants are usable, including some for controls that have no Unicode names:

```
name                                   character

END OF PROTECTED AREA                  END OF GUARDED AREA, U+0097
HIGH OCTET PRESET                      U+0081
HOP                                    U+0081
IND                                    U+0084
INDEX                                  U+0084
PAD                                    U+0080
PADDING CHARACTER                      U+0080
PRIVATE USE 1                          PRIVATE USE ONE, U+0091
PRIVATE USE 2                          PRIVATE USE TWO, U+0092
SGC                                    U+0099
SINGLE GRAPHIC CHARACTER INTRODUCER    U+0099
SINGLE-SHIFT 2                         SINGLE SHIFT TWO, U+008E
SINGLE-SHIFT 3                         SINGLE SHIFT THREE, U+008F
START OF PROTECTED AREA                START OF GUARDED AREA, U+0096
```

CUSTOM ALIASES

You can add customized aliases to standard (`:full`) Unicode naming conventions. The aliases override any standard definitions, so, if you're twisted enough, you can change `"\N{LATIN CAPITAL LETTER A}"` to mean `"B"`, etc.

Note that an alias should not be something that is a legal curly brace-enclosed quantifier (see *"QUANTIFIERS" in perlreref*). For example `\N{123}` means to match 123 non-newline characters, and is not treated as a charnames alias. Aliases are discouraged from beginning with anything other than an alphabetic character and from containing anything other than alphanumerics, spaces, dashes, parentheses, and underscores. Currently they must be ASCII.

An alias can map to either an official Unicode character name or to a numeric code point (ordinal). The latter is useful for assigning names to code points in Unicode private use areas such as U+E800 through U+F8FF. A numeric code point must be a non-negative integer or a string beginning with `"U+"` or `"0x"` with the remainder considered to be a hexadecimal integer. A literal numeric constant must be unsigned; it will be interpreted as hex if it has a leading zero or contains non-decimal hex digits; otherwise it will be interpreted as decimal.

Aliases are added either by the use of anonymous hashes:

```
use charnames ":alias" => {
    e_ACUTE => "LATIN SMALL LETTER E WITH ACUTE",
    mychar1 => 0xE8000,
    };
my $str = "\N{e_ACUTE}";
```

or by using a file containing aliases:

```
use charnames ":alias" => "pro";
```

This will try to read `"unicore/pro_alias.pl"` from the `@INC` path. This file should return a list in plain perl:

```
(
A_GRAVE         => "LATIN CAPITAL LETTER A WITH GRAVE",
A_CIRCUM        => "LATIN CAPITAL LETTER A WITH CIRCUMFLEX",
A_DIAERES       => "LATIN CAPITAL LETTER A WITH DIAERESIS",
A_TILDE         => "LATIN CAPITAL LETTER A WITH TILDE",
A_BREVE         => "LATIN CAPITAL LETTER A WITH BREVE",
A_RING          => "LATIN CAPITAL LETTER A WITH RING ABOVE",
A_MACRON        => "LATIN CAPITAL LETTER A WITH MACRON",
mychar2         => "U+E8001",
);
```

Both these methods insert `":full"` automatically as the first argument (if no other argument is given), and you can give the `":full"` explicitly as well, like

```
    use charnames ":full", ":alias" => "pro";
```

Also, both these methods currently allow only a single character to be named. To name a sequence of characters, use a *custom translator* (described below).

charnames::viacode(code)

Returns the full name of the character indicated by the numeric code. For example,

```
    print charnames::viacode(0x2722);
```

prints "FOUR TEARDROP-SPOKED ASTERISK".

The name returned is the official name for the code point, if available; otherwise your custom alias for it. This means that your alias will only be returned for code points that don't have an official Unicode name (nor Unicode version 1 name), such as private use code points, and the 4 control characters U+0080, U+0081, U+0084, and U+0099. If you define more than one name for the code point, it is indeterminate which one will be returned.

The function returns `undef` if no name is known for the code point. In Unicode the proper name of these is the empty string, which `undef` stringifies to. (If you ask for a code point past the legal Unicode maximum of U+10FFFF that you haven't assigned an alias to, you get `undef` plus a warning.)

The input number must be a non-negative integer or a string beginning with `"U+"` or `"0x"` with the remainder considered to be a hexadecimal integer. A literal numeric constant must be unsigned; it will be interpreted as hex if it has a leading zero or contains non-decimal hex digits; otherwise it will be interpreted as decimal.

Notice that the name returned for of U+FEFF is "ZERO WIDTH NO-BREAK SPACE", not "BYTE ORDER MARK".

charnames::string_vianame(name)

This is a runtime equivalent to `\N{...}`. *name* can be any expression that evaluates to a name accepted by `\N{...}` under the *`:full` option* to `charnames`. In addition, any other options for the controlling `"use charnames"` in the same scope apply, like any *script list, `:short` option*, or *custom aliases* you may have defined.

The only difference is that if the input name is unknown, `string_vianame` returns `undef` instead of the REPLACEMENT CHARACTER and does not raise a warning message.

charnames::vianame(name)

This is similar to `string_vianame`. The main difference is that under most circumstances (see *BUGS* for the others), vianame returns an ordinal code point, whereas `string_vianame` returns a string. For example,

```
   printf "U+%04X", charnames::vianame("FOUR TEARDROP-SPOKED ASTERISK");
```

prints "U+2722".

This leads to the other two differences. Since a single code point is returned, the function can't handle named character sequences, as these are composed of multiple characters. And, the code point can be that of any character, even ones that aren't legal under the `use bytes` pragma,

CUSTOM TRANSLATORS

The mechanism of translation of `\N{...}` escapes is general and not hardwired into *charnames.pm*. A module can install custom translations (inside the scope which `use`s the module) with the following magic incantation:

```
    sub import {
        shift;
        $^H{charnames} = \&translator;
    }
```

Here translator() is a subroutine which takes *CHARNAME* as an argument, and returns text to insert into the string instead of the `\N{`*CHARNAME*`}` escape. Since the text to insert should be different in `bytes` mode and out of it, the function should check the current state of `bytes`-flag as in:

```
    use bytes ();                      # for $bytes::hint_bits
    sub translator {
        if ($^H & $bytes::hint_bits) {
            return bytes_translator(@_);
        }
        else {
            return utf8_translator(@_);
        }
    }
```

See *CUSTOM ALIASES* above for restrictions on *CHARNAME*.

Of course, `vianame` and `viacode` would need to be overridden as well.

BUGS

vianame normally returns an ordinal code point, but when the input name is of the form `U+...`, it returns a chr instead. In this case, if `use bytes` is in effect and the character won't fit into a byte, it returns `undef` and raises a warning.

Names must be ASCII characters only, which means that you are out of luck if you want to create aliases in a language where some or all the characters of the desired aliases are non-ASCII.

Since evaluation of the translation function (see *CUSTOM TRANSLATORS*) happens in the middle of compilation (of a string literal), the translation function should not do any `eval`s or `require`s. This restriction should be lifted (but is low priority) in a future version of Perl.