NAME

Unicode::UCD - Unicode character database

SYNOPSIS

```
use Unicode::UCD 'charinfo';
my $charinfo   = charinfo($codepoint);

use Unicode::UCD 'charblock';
my $charblock  = charblock($codepoint);

use Unicode::UCD 'charscript';
my $charscript = charscript($codepoint);

use Unicode::UCD 'charblocks';
my $charblocks = charblocks();

use Unicode::UCD 'charscripts';
my %charscripts = charscripts();

use Unicode::UCD qw(charscript charinrange);
my $range = charscript($script);
print "looks like $script\n" if charinrange($range, $codepoint);

use Unicode::UCD 'compexcl';
my $compexcl = compexcl($codepoint);

use Unicode::UCD 'namedseq';
my $namedseq = namedseq($named_sequence_name);

my $unicode_version = Unicode::UCD::UnicodeVersion();
```

DESCRIPTION

The Unicode::UCD module offers a simple interface to the Unicode Character Database.

charinfo

```
use Unicode::UCD 'charinfo';

my $charinfo = charinfo(0x41);
```

charinfo() returns a reference to a hash that has the following fields as defined by the Unicode standard:

```
key

code                code point with at least four hexdigits
name                name of the character IN UPPER CASE
category            general category of the character
combining           classes used in the Canonical Ordering Algorithm
bidi                bidirectional category
decomposition       character decomposition mapping
decimal             if decimal digit this is the integer numeric value
digit               if digit this is the numeric value
```

```
    numeric         if numeric is the integer or rational numeric value
    mirrored        if mirrored in bidirectional text
    unicode10       Unicode 1.0 name if existed and different
    comment         ISO 10646 comment field
    upper           uppercase equivalent mapping
    lower           lowercase equivalent mapping
    title           titlecase equivalent mapping

    block           block the character belongs to (used in \p{In...})
    script          script the character belongs to
```

If no match is found, a reference to an empty hash is returned.

The `block` property is the same as returned by charinfo(). It is not defined in the Unicode Character Database proper (Chapter 4 of the Unicode 3.0 Standard, aka TUS3) but instead in an auxiliary database (Chapter 14 of TUS3). Similarly for the `script` property.

Note that you cannot do (de)composition and casing based solely on the above `decomposition` and `lower`, `upper`, `title`, properties, you will need also the compexcl(), casefold(), and casespec() functions.

charblock

```
    use Unicode::UCD 'charblock';

    my $charblock = charblock(0x41);
    my $charblock = charblock(1234);
    my $charblock = charblock("0x263a");
    my $charblock = charblock("U+263a");

    my $range     = charblock('Armenian');
```

With a **code point argument** charblock() returns the *block* the character belongs to, e.g. `Basic Latin`. Note that not all the character positions within all blocks are defined.

See also *Blocks versus Scripts*.

If supplied with an argument that can't be a code point, charblock() tries to do the opposite and interpret the argument as a character block. The return value is a *range*: an anonymous list of lists that contain *start-of-range*, *end-of-range* code point pairs. You can test whether a code point is in a range using the *charinrange* function. If the argument is not a known character block, `undef` is returned.

charscript

```
    use Unicode::UCD 'charscript';

    my $charscript = charscript(0x41);
    my $charscript = charscript(1234);
    my $charscript = charscript("U+263a");

    my $range      = charscript('Thai');
```

With a **code point argument** charscript() returns the *script* the character belongs to, e.g. `Latin`, `Greek`, `Han`.

See also *Blocks versus Scripts*.

If supplied with an argument that can't be a code point, charscript() tries to do the opposite and interpret the argument as a character script. The return value is a *range*: an anonymous list of lists that contain *start-of-range*, *end-of-range* code point pairs. You can test whether a code point is in a range using the *charinrange* function. If the argument is not a known character script, `undef` is returned.

charblocks

```
    use Unicode::UCD 'charblocks';

    my $charblocks = charblocks();
```

charblocks() returns a reference to a hash with the known block names as the keys, and the code point ranges (see *charblock*) as the values.

See also *Blocks versus Scripts*.

charscripts

```
    use Unicode::UCD 'charscripts';

    my %charscripts = charscripts();
```

charscripts() returns a hash with the known script names as the keys, and the code point ranges (see *charscript*) as the values.

See also *Blocks versus Scripts*.

Blocks versus Scripts

The difference between a block and a script is that scripts are closer to the linguistic notion of a set of characters required to present languages, while block is more of an artifact of the Unicode character numbering and separation into blocks of (mostly) 256 characters.

For example the Latin **script** is spread over several **blocks**, such as `Basic Latin`, `Latin 1 Supplement`, `Latin Extended-A`, and `Latin Extended-B`. On the other hand, the Latin script does not contain all the characters of the `Basic Latin` block (also known as the ASCII): it includes only the letters, and not, for example, the digits or the punctuation.

For blocks see http://www.unicode.org/Public/UNIDATA/Blocks.txt

For scripts see UTR #24: http://www.unicode.org/unicode/reports/tr24/

Matching Scripts and Blocks

Scripts are matched with the regular-expression construct `\p{...}` (e.g. `\p{Tibetan}` matches characters of the Tibetan script), while `\p{In...}` is used for blocks (e.g. `\p{InTibetan}` matches any of the 256 code points in the Tibetan block).

Code Point Arguments

A *code point argument* is either a decimal or a hexadecimal scalar designating a Unicode character, or `U+` followed by hexadecimals designating a Unicode character. In other words, if you want a code point to be interpreted as a hexadecimal number, you must prefix it with either `0x` or `U+`, because a string like e.g. `123` will be interpreted as a decimal code point. Also note that Unicode is **not** limited to 16 bits (the number of Unicode characters is open-ended, in theory unlimited): you may have more than 4 hexdigits.

charinrange

In addition to using the `\p{In...}` and `\P{In...}` constructs, you can also test whether a code point is in the *range* as returned by *charblock* and *charscript* or as the values of the hash returned by *charblocks* and *charscripts* by using charinrange():

```
use Unicode::UCD qw(charscript charinrange);

$range = charscript('Hiragana');
print "looks like hiragana\n" if charinrange($range, $codepoint);
```

compexcl

```
use Unicode::UCD 'compexcl';

my $compexcl = compexcl("09dc");
```

The compexcl() returns the composition exclusion (that is, if the character should not be produced during a precomposition) of the character specified by a **code point argument**.

If there is a composition exclusion for the character, true is returned. Otherwise, false is returned.

casefold

```
use Unicode::UCD 'casefold';

my $casefold = casefold("00DF");
```

The casefold() returns the locale-independent case folding of the character specified by a **code point argument**.

If there is a case folding for that character, a reference to a hash with the following fields is returned:

```
key

code                code point with at least four hexdigits
status              "C", "F", "S", or "I"
mapping             one or more codes separated by spaces
```

The meaning of the *status* is as follows:

```
C                   common case folding, common mappings shared
                    by both simple and full mappings
F                   full case folding, mappings that cause strings
                    to grow in length. Multiple characters are separated
                    by spaces
S                   simple case folding, mappings to single characters
                    where different from F
I                   special case for dotted uppercase I and
                    dotless lowercase i
                    - If this mapping is included, the result is
                      case-insensitive, but dotless and dotted I's
                      are not distinguished
                    - If this mapping is excluded, the result is not
                      fully case-insensitive, but dotless and dotted
                      I's are distinguished
```

If there is no case folding for that character, `undef` is returned.

For more information about case mappings see http://www.unicode.org/unicode/reports/tr21/

casespec

```
use Unicode::UCD 'casespec';

my $casespec = casespec("FB00");
```

The casespec() returns the potentially locale-dependent case mapping of the character specified by a **code point argument**. The mapping may change the length of the string (which the basic Unicode case mappings as returned by charinfo() never do).

If there is a case folding for that character, a reference to a hash with the following fields is returned:

```
key

code             code point with at least four hexdigits
lower            lowercase
title            titlecase
upper            uppercase
condition        condition list (may be undef)
```

The `condition` is optional. Where present, it consists of one or more *locales* or *contexts*, separated by spaces (other than as used to separate elements, spaces are to be ignored). A condition list overrides the normal behavior if all of the listed conditions are true. Case distinctions in the condition list are not significant. Conditions preceded by "NON_" represent the negation of the condition.

Note that when there are multiple case folding definitions for a single code point because of different locales, the value returned by casespec() is a hash reference which has the locales as the keys and hash references as described above as the values.

A *locale* is defined as a 2-letter ISO 3166 country code, possibly followed by a "_" and a 2-letter ISO language code (possibly followed by a "_" and a variant code). You can find the lists of those codes, see *Locale::Country* and *Locale::Language*.

A *context* is one of the following choices:

```
FINAL            The letter is not followed by a letter of
                 general category L (e.g. Ll, Lt, Lu, Lm, or Lo)
MODERN           The mapping is only used for modern text
AFTER_i          The last base character was "i" (U+0069)
```

For more information about case mappings see http://www.unicode.org/unicode/reports/tr21/

namedseq()

```
use Unicode::UCD 'namedseq';

my $namedseq = namedseq("KATAKANA LETTER AINU P");
my @namedseq = namedseq("KATAKANA LETTER AINU P");
my %namedseq = namedseq();
```

If used with a single argument in a scalar context, returns the string consisting of the code points of the named sequence, or `undef` if no named sequence by that name exists. If used with a single argument in a list context, returns list of the code points. If used with no arguments in a list context, returns a hash with the names of the named sequences as the keys and the named sequences as strings as the values. Otherwise, returns `undef` or empty list depending on the context.

(New from Unicode 4.1.0)

Unicode::UCD::UnicodeVersion

Unicode::UCD::UnicodeVersion() returns the version of the Unicode Character Database, in other words, the version of the Unicode standard the database implements. The version is a string of numbers delimited by dots (`'.'`).

Implementation Note

The first use of charinfo() opens a read-only filehandle to the Unicode Character Database (the database is included in the Perl distribution). The filehandle is then kept open for further queries. In other words, if you are wondering where one of your filehandles went, that's where.

BUGS

Does not yet support EBCDIC platforms.

AUTHOR

Jarkko Hietaniemi