UTF : An Unicode string manager

The UTF class is an utility class that eases the handling of Unicode strings.

Namespace: `\`
File location: `lib/utf.php`

Instantiation

Return class instance

```
$utf = \UTF::instance();
```

The UTF class uses the Prefab (prefab-registry) factory wrapper, so you can grab the same instance of that class at any point of your code.

Methods

Similarly to the standard PHP strings methods, all the methods of the UTF class return zero-based offsets.

strlen

Get string length

```
int strlen ( string $str )
```

This function returns the length of a given string

Example:

```
$utf->strlen('나는 유리를 먹을 수 있어요. 그래도'); // returns 20  (while php strlen returns 48)
```

stripos

Find position of first occurrence of a string, case-insensitive

```
int|FALSE stripos ( string $stack, string $needle [, int $ofs = 0 ] )
```

This function returns the position of the first occurrence of the `$needle` string in the `$stack` string. Case-insensitive search. Returns `FALSE` if `$needle` not found.

If `$ofs` is specified, search will start this number of characters counted from the beginning of the string. The `$ofs` offset cannot be negative.

Examples:

```
$utf->stripos('Les Naïfs ægithales hâtifs', 'naïfs');     // returns 4
$utf->stripos('Les Naïfs ægithales hâtifs', 'NAÏFS');     // returns 4
$utf->stripos('Les Naïfs ægithales hâtifs', 'NAÏFS', 10); // returns FALSE
```

strpos

Find position of first occurrence of a string, case-sensitive

```
int|FALSE strpos ( string $stack, string $needle [, int $ofs = 0 [, bool $case = FALS
E ]] )
```

This function returns the position of the first occurrence of the `$needle` string in the `$stack` string. Returns `FALSE` if `$needle` not found.

If `$ofs` is specified, search will start this number of characters counted from the beginning of the string. The `$ofs` offset cannot be negative.

If `$case` is set to `TRUE`, the search is case-insensitive and the function behaves like `stripos()` (utf-unicode-string-manager#stripos).

Examples:

```
$utf->strpos('Góa ē-tàng Chiảh Po-lê', 'Góa' );     // returns 0
$utf->strpos('Góa ē-tàng Chiảh Po-lê', 'Góa', 4 );  // returns FALSE
$utf->strpos('Góa ē-tàng Chiảh Po-lê', 'chiảh', 0 );       // returns FALSE  (case-se
nsitive)
$utf->strpos('Góa ē-tàng Chiảh Po-lê', 'chiảh', 0, TRUE ); // returns 11     (case-in
sensitive)
```

stristr

Returns part of haystack string from the first occurrence of the needle to the end of haystack, case-insensitive

```
string|FALSE stristr ( string $stack, string $needle [, bool $before = FALSE ] )
```

This function returns part of `$stack` string starting from and including the first occurrence of `$needle` to the end of `$stack`. Case-insensitive. Returns `FALSE` if `$needle` not found.

If `$before` is set to `TRUE`, stristr() returns the part of the `$stack` before the first occurrence of the `$needle` (excluding the needle).

Examples:

```
$utf->stristr('Mayia Góa Chàyiảh Lêh-Pok', 'CHÀYIẢH' );   // returns 'Chàyiảh Lêh-Po
k'
$utf->stristr('Mayia Góa Chàyiảh Lêh-Pok', 'GóA', TRUE ); // returns 'Mayia '
```

strstr

Returns part of haystack string from the first occurrence of the needle to the end of the haystack

```
string|FALSE strstr ( string $stack, string $needle [, bool $before = FALSE [, bool
$case = FALSE ]] )
```

This function returns part of `$stack` string starting from and including the first occurrence of `$needle` to the end of `$stack` . Returns `FALSE` if `$needle` not found.

If `$before` is set to `TRUE` , strstr() returns the part of the `$stack` before the first occurrence of the `$needle` (excluding the needle).

If `$case` is set to `TRUE` , the search is case-insensitive and the function behaves like `stristr()` (utf-unicode-string-manager#stristr).

Example:

```
$email  = 'Mïchaño@example.com';
$domain = $utf->strstr($email, '@');  // returns '@example.com'

$user   = $utf->strstr($email, '@', TRUE);  // returns 'Mïchaño'
```

substr

Return part of a string

```
string|FALSE substr ( string $str, int $start [, int $length = 0 ] )
```

This function returns the portion of string `$str` specified by the `$start` and `$length` parameters. If `$length` is omitted, the substring starting from `$start` until the end of the string will be returned.

If `$start` is negative, the returned string will `$start` at the start'th character from the end of string `$str` . If string `$str` is less than or equal to `$start` characters long, `FALSE` will be returned.

Examples:

```
$utf->substr('El pingüino Wenceslao hizo kilómetros',  3,8); // returns 'pingüino'
$utf->substr('El pingüino Wenceslao hizo kilómetros',-10,4); // returns 'kiló'
```

substr_count

Count the number of occurrences of a substring

```
int substr_count ( string $stack, string $needle )
```

This function counts and returns the number of times the `$needle` substring occurs in the `$stack` string. Note that `$needle` is case sensitive.

Examples:

```
$utf->substr_count('This is an example as it is', 'is'); // returns 3
$utf->substr_count(implode(array('This','example','as','it')), 'is'); // returns 1 !
PHP BUG !
$arr = array('This','example','as','it');
$utf->substr_count(implode($arr,'is'), 'is'); // returns 4
```

ltrim

Strip whitespaces from the beginning of a string

```
string ltrim ( string $str )
```

This function strips whitespaces and other characters (according to the regexp `/[\pZ\pC]+/u`) from the beginning of a given string.

Examples:

```
$utf->ltrim("\xe2\x80\x83\x20  WhatAMana!\xc2\xa0\xe1\x9a\x80"); // returns "WhatAMan
a!\xc2\xa0\xe1\x9a\x80"
$utf->ltrim('   invisible leading spaces...   '); // returns 'invisible leading space
s...   '
```

rtrim

Strip whitespaces from the end of a string

```
string rtrim ( string $str )
```

This function strips whitespaces and other characters (according to the regexp `/[\pZ\pC]+$/u`) from the end of a given string.

Examples:

```
$utf->rtrim("\xe2\x80\x83\x20  WhatAMana!   \xc2\xa0\xe1\x9a\x80"); // returns "\xe2
\x80\x83\x20  WhatAMana!"
$utf->rtrim('   invisible trailing spaces...      '); // returns '   invisible traili
ng spaces...'
```

trim

Strip whitespaces from the beginning and end of a string

```
string trim ( string $str )
```

This function strips whitespaces and other characters (according to the regexp `/^[\pZ\pC]+|[\pZ\pC]+$/u`) from the beginning and end of a given string.

Examples:

```
$utf->trim("\xe2\x80\x83\x20  WhatAMana!  \xc2\xa0\xe1\x9a\x80"); // returns "WhatAMa
na!"
$utf->trim('   invisible spaces...      '); // returns 'invisible spaces...'
```

bom

Return UTF-8 byte order mark (BOM (https://en.wikipedia.org/wiki/Byte_order_mark))

```
string bom (  )
```

Return the byte order mark (BOM (https://en.wikipedia.org/wiki/Byte_order_mark)) Unicode character used to signal the byte order of a text file or a stream. The BOM character may also indicate which of the several Unicode representations the text is encoded in. BOM use is optional, and, if used, must appear at the start of the text stream.

Example:

```
$bom = \UTF::instance()->bom();  // $bom = 0xefbbbf
echo '0x'.dechex(ord($bom[0])).dechex(ord($bom[1])).dechex(ord($bom[2])); // displays
'0xefbbbf'

// convert/save a file with a BOM at its beginning
$f3->write( $filename, $bom . $f3->read($filename) );
```

translate

Convert code points to Unicode symbols

```
string translate ( string $str )
```

Converts and returns code points (https://en.wikipedia.org/wiki/Code_point) (e.g. U+0E8D U+053D) to the equivalent Unicode symbols (e.g. ຍ Ս)

emojify

Translate emoji tokens to Unicode font-supported symbols

```
string emojify ( string $str )
```

Converts and returns the Unicode font-supported symbols equivalent of emoji tokens.

emoji tokens translated by default are:

```
':(' => '\u2639', // frown
':)' => '\u263a', // smile
'<3' => '\u2665', // heart
':D' => '\u1f603', // grin
'XD' => '\u1f606', // laugh
';)' => '\u1f609', // wink
':P' => '\u1f60b', // tongue
':,' => '\u1f60f', // think
':/' => '\u1f623', // skeptic
'8O' => '\u1f632', // oops
```

Example:

```
echo \UTF::instance()->emojify('Thanks :) I <3');  //  displays 'Thanks ☺ I ♥'
```

> Keep in mind you need a font that supports these characters to even have a hope of seeing them correctly in your browser pages.

You can specify your own additional emoji tokens with the EMOJI system variable (quick-reference#emoji). When provided, those emoji tokens are added to the basic set above and will be used when translating a string to Unicode font-supported symbols.

Examples:

```
$f3->set('EMOJI', array('(c)' => '&#169;', '?' => '&#191') );
echo \UTF::instance()->emojify( 'Do you like (c)opyrights ??');  //  displays 'Do you
like ©opyrights ¿¿'
//
$f3->set('EMOJI', array('@om' => '\U0F00', '&ooooooom' => '\U0F02', '%om' => '\U0F00')
);
echo \UTF::instance()->emojify( '@om Greets &ooooooom from Tibet %om');  //  displays
'ॐ Greets ॐ from Tibet ॐ'
```

As you can see, it's up to you to define your emoji tokens as you fancy. You can even imagine to automate the call to the `emojify` function for the variables you use in your templates.