Internationalization Functions

Introduction

Internationalization extension (further is referred as Intl) is a wrapper for » ICU library, enabling PHP programmers to perform UCA-conformant collation and date/time/number/currency formatting in their scripts.

It tends to closely follow ICU APIs, so that people having experience working with ICU in either C/C++ or Java could easily use the PHP API. Also, this way ICU documentation would be useful to understand various ICU functions.

Intl consists of several modules, each of them exposes the corresponding ICU API:

- Collator: provides string comparison capability with support for appropriate locale-sensitive sort orderings.
- Number Formatter: allows to display number according to the localized format or given pattern or set of rules, and to parse strings into numbers.
- Message Formatter: allows to create messages incorporating data (such as numbers or dates) formatted according to given pattern and locale rules, and parse messages extracting data from them.
- Normalizer: provides a function to transform text into one of the Unicode normalization forms, and provides a routine to test if a given string is already normalized.
- Locale: provides interaction with locale identifiers in the form of functions to get subtags from locale identifier; parse, compose, match(lookup and filter) locale identifiers.

Links

- » Miscellaneous ICU docs
- » ICU User Guide
- » Unicode Collation Algorithm

Installing/Configuring

Requirements

To build the extension you need to install the » ICU library of version 3.6+.

You will also need the latest version of PHP. Collator is known to work well on PHP 5.1.3+ and 5.2.0+.

Installation

Run:
```
$ make install
```

Then enable the extension by adding the following line to [PHP] section of your *php.ini*:
```
extension=intl.so
```

Testing

Run:
```
$ make test
```

Note that the tests may fail if:

- The Collator extension is already enabled in *php.ini*
- LD_LIBRARY_PATH is used to load ICU libraries and value of the 'variables_order' setting in php.ini doesn't contain letter 'E' (missing 'E' means "do not register Environment variables");

Building

Let's assume that you have installed PHP to */opt/php5/* and ICU is installed to */opt/icu/*. Run the following commands:

```
$ /opt/php5/bin/phpize
$ ./configure  --with-php-config=/opt/php5/bin/php-config
--with-icu-dir=/opt/icu
$ make
```

If your ICU is installed to a non-standard directory then you might want to specify its location in **LD_LIBRARY_PATH** environment variable so that dynamic linker can find it:

```
$ export LD_LIBRARY_PATH=/opt/icu/lib
```

Otherwise, if PHP and ICU are installed to their default locations, then the additional options to `configure' are not needed.

Resource Types

This extension has no resource types defined.

Predefined Constants

INTL_MAX_LOCALE_LEN (integer)
Limit on locale length, set to 64 in PHP code. Locale names longer than this limit will not be accepted.

Examples

Basic usage of this extension

Each module provides two kind of APIs: a procedural one and an object oriented one. Both are actually identical and described in the corresponding document.

Note

All input strings must be in UTF-8 encoding. All output strings are also in UTF-8.

Example #1 - Example of using the procedural API

```
<?php
$coll  = collator_create('en_US');
$result = collator_compare($coll, "string#1", "string#2");
?>
```

Example #2 - Example of using the object-oriented API

```
<?php
$coll = new Collator('en_US');
$al   = $coll->getLocale(Locale::ACTUAL_LOCALE);
echo "Actual locale: $al\n";

$formatter = new NumberFormatter('en_US', NumberFormatter::DECIMAL);
echo $formatter->format(1234567);
?>
```

intl Functions

grapheme_extract

grapheme_extract -- Function to extract a sequence of default grapheme clusters from a text buffer, which must be encoded in UTF-8.

Description

Procedural style

string **grapheme_extract** (string `$haystack`, int `$size` [, int `$extract_type` [, int `$start` [, int `&$next`]]])

Function to extract a sequence of default grapheme clusters from a text buffer, which must be encoded in UTF-8.

Parameters

`haystack`
: String to search.

`size`
: Maximum number items - based on the $extract_type - to return.

`extract_type`
: Defines the type of units referred to by the $size parameter:

- GRAPHEME_EXTR_COUNT (default) - $size is the number of default grapheme clusters to extract.
- GRAPHEME_EXTR_MAXBYTES - $size is the maximum number of bytes returned.
- GRAPHEME_EXTR_MAXCHARS - $size is the maximum number of UTF-8 characters returned.

`start`
: Starting position in $haystack in bytes - if given, it must be zero or a positive value that is less than or equal to the length of $haystack in bytes. The default is zero. If $start does not point to the first byte of a UTF-8 character, the start position is moved to the next character boundary.

`next`
: Reference to a value that will be set to the next starting position. When the call returns, this may point to the first byte position past the end of the string.

Return Values

A string starting at offset $start and ending on a default grapheme cluster boundary that

conforms to the $size and $extract_type specified.

Examples

Example #3 - grapheme_extract() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"

print urlencode(grapheme_extract( $char_a_ring_nfd . $char_o_diaeresis_nfd,
1, GRAPHEME_EXTR_COUNT, 2));

?>
```

The above example will output:

```
o%CC%88
```

See Also

- grapheme_substr()
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_stripos

grapheme_stripos -- Find position (in grapheme units) of first occurrence of a case-insensitive string

Description

Procedural style

int **grapheme_stripos** (string `$haystack`, string `$needle` [, int `$offset`])

Find position (in grapheme units) of first occurrence of a case-insensitive string

Parameters

`haystack`
: The string to look in. Must be valid UTF-8.

`needle`
: The string to look for. Must be valid UTF-8.

`offset`
: The optional $offset parameter allows you to specify where in haystack to start searching as an offset in grapheme units (not bytes or characters). If not given, the default is zero. The position returned is still relative to the beginning of haystack regardless of the value of $offset.

Return Values

Returns the position as an integer. If needle is not found, grapheme_stripos() will return boolean FALSE.

Examples

Example #4 - grapheme_stripos() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"
$char_O_diaeresis_nfd = "O\xCC\x88"; // 'LATIN CAPITAL LETTER O WITH
DIAERESIS' (U+00D6) normalization form "D"

print grapheme_stripos( $char_a_ring_nfd . $char_a_ring_nfd .
$char_o_diaeresis_nfd, $char_O_diaeresis_nfd);
```

```
?>
```

The above example will output:

```
2
```

See Also

- grapheme_stristr()
- grapheme_strpos()
- grapheme_strripos()
- grapheme_strrpos()
- grapheme_strstr()
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_stristr

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

Description

Procedural style

string **grapheme_stristr** (string `$haystack`, string `$needle` [, boolean `$before_needle`])

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

Parameters

`haystack`
: The input string. Must be valid UTF-8.

`needle`
: The string to look for. Must be valid UTF-8.

`before_needle`
: If TRUE (the default is FALSE), grapheme_strstr() returns the part of the haystack before the first occurence of the needle.

Return Values

Returns the portion of $haystack, or FALSE if $needle is not found.

Examples

Example #5 - grapheme_stristr() **example**

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"
$char_O_diaeresis_nfd = "O\xCC\x88"; // 'LATIN CAPITAL LETTER O WITH
DIAERESIS' (U+00D6) normalization form "D"

print urlencode(grapheme_stristr( $char_a_ring_nfd . $char_o_diaeresis_nfd .
$char_a_ring_nfd, $char_O_diaeresis_nfd));

?>
```

The above example will output:

```
o%CC%88a%CC%8A
```

See Also

- grapheme_stripos()
- grapheme_strpos()
- grapheme_strripos()
- grapheme_strrpos()
- grapheme_strstr()
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_strlen

grapheme_strlen -- Get string length in grapheme units

Description

Procedural style

int **grapheme_strlen** (string `$input`)

Get string length in grapheme units (not bytes or characters)

Parameters

`input`
The string being measured for length. It must be a valid UTF-8 string.

Return Values

The length of the string on success, and 0 if the string is empty.

Examples

Example #6 - grapheme_strlen() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"

print grapheme_strlen( 'abc' . $char_a_ring_nfd . $char_o_diaeresis_nfd .
$char_a_ring_nfd);

?>
```

The above example will output:

```
6
```

See Also

- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_strpos

grapheme_strpos -- Find position (in grapheme units) of first occurrence of a string

Description

Procedural style

int **grapheme_strpos** (string `$haystack`, string `$needle` [, int `$offset`])

Find position (in grapheme units) of first occurrence of a string

Parameters

`haystack`
: The string to look in. Must be valid UTF-8.

`needle`
: The string to look for. Must be valid UTF-8.

`offset`
: The optional $offset parameter allows you to specify where in $haystack to start searching as an offset in grapheme units (not bytes or characters). If not given, the default is zero. The position returned is still relative to the beginning of haystack regardless of the value of $offset.

Return Values

Returns the position as an integer. If needle is not found, strpos() will return boolean FALSE.

Examples

Example #7 - grapheme_strpos() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"

print grapheme_strpos( $char_a_ring_nfd . $char_a_ring_nfd .
$char_o_diaeresis_nfd, $char_o_diaeresis_nfd);

?>
```

The above example will output:

```
2
```

See Also

- grapheme_stripos()
- grapheme_stristr()
- grapheme_strripos()
- grapheme_strrpos()
- grapheme_strstr()
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_strripos

grapheme_strripos -- Find position (in grapheme units) of last occurrence of a case-insensitive string

Description

Procedural style

int **grapheme_strripos** (string `$haystack`, string `$needle` [, int `$offset`])

Find position (in grapheme units) of last occurrence of a case-insensitive string

Parameters

`haystack`
: The string to look in. Must be valid UTF-8.

`needle`
: The string to look for. Must be valid UTF-8.

`offset`
: The optional $offset parameter allows you to specify where in $haystack to start searching as an offset in grapheme units (not bytes or characters). If not given, the default is zero. The position returned is still relative to the beginning of haystack regardless of the value of $offset.

Return Values

Returns the position as an integer. If needle is not found, grapheme_strripos() will return boolean FALSE.

Examples

Example #8 - grapheme_strripos() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"
$char_O_diaeresis_nfd = "O\xCC\x88"; // 'LATIN CAPITAL LETTER O WITH
DIAERESIS' (U+00D6) normalization form "D"

print grapheme_strripos( $char_a_ring_nfd . $char_o_diaeresis_nfd .
$char_o_diaeresis_nfd, $char_O_diaeresis_nfd);
```

```
?>
```

The above example will output:

```
2
```

See Also

- grapheme_stripos()
- grapheme_stristr()
- grapheme_strpos()
- grapheme_strrpos()
- grapheme_strstr()
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_strrpos

grapheme_strrpos -- Find position (in grapheme units) of last occurrence of a string

Description

Procedural style

int **grapheme_strrpos** (string `$haystack`, string `$needle` [, int `$offset`])

Find position (in grapheme units) of last occurrence of a string

Parameters

`haystack`
: The string to look in. Must be valid UTF-8.

`needle`
: The string to look for. Must be valid UTF-8.

`offset`
: The optional $offset parameter allows you to specify where in $haystack to start searching as an offset in grapheme units (not bytes or characters). If not given, the default is zero. The position returned is still relative to the beginning of haystack regardless of the value of $offset.

Return Values

Returns the position as an integer. If needle is not found, grapheme_strrpos() will return boolean FALSE.

Examples

Example #9 - grapheme_strrpos() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"

print grapheme_strrpos( $char_a_ring_nfd . $char_o_diaeresis_nfd .
$char_o_diaeresis_nfd, $char_o_diaeresis_nfd);

?>
```

The above example will output:

```
2
```

See Also

- grapheme_stripos()
- grapheme_stristr()
- grapheme_strpos()
- grapheme_strripos()
- grapheme_strstr()
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_strstr

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

Description

Procedural style

string **grapheme_strstr** (string `$haystack`, string `$needle` [, boolean `$before_needle`])

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

Parameters

`haystack`
: The input string. Must be valid UTF-8.

`needle`
: The string to look for. Must be valid UTF-8.

`before_needle`
: If TRUE (the default is FALSE), grapheme_strstr() returns the part of the haystack before the first occurence of the needle.

Return Values

Returns the portion of string, or FALSE if needle is not found.

Examples

Example #10 - grapheme_strstr() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"

print urlencode(grapheme_stristr( $char_a_ring_nfd . $char_o_diaeresis_nfd .
$char_a_ring_nfd, $char_o_diaeresis_nfd));

?>
```

The above example will output:

o%CC%88a%CC%8A

See Also

- grapheme_stristr()
- grapheme_stripos()
- grapheme_strpos()
- grapheme_strripos()
- grapheme_strrpos()
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

grapheme_substr

grapheme_substr -- Return part of a string

Description

Procedural style

int **grapheme_substr** (string `$string`, int `$start` [, int `$length`])

Return part of a string

Parameters

`string`
The input string. Must be valid UTF-8.

`start`
Start position in default grapheme units. If $start is non-negative, the returned string will start at the $start'th position in $string, counting from zero. If $start is negative, the returned string will start at the $start'th grapheme unit from the end of string.

`length`
Length in grapheme units. If $length is given and is positive, the string returned will contain at most $length grapheme units beginning from $start (depending on the length of string). If $string is less than or equal to $start grapheme units long, FALSE will be returned. If $length is given and is negative, then that many grapheme units will be omitted from the end of string (after the start position has been calculated when a start is negative). If $start denotes a position beyond this truncation, an empty string will be returned.

Return Values

Returns the extracted part of $string.

Examples

Example #11 - grapheme_substr() example

```
< ?php

$char_a_ring_nfd = "a\xCC\x8A";  // 'LATIN SMALL LETTER A WITH RING ABOVE'
(U+00E5) normalization form "D"
$char_o_diaeresis_nfd = "o\xCC\x88"; // 'LATIN SMALL LETTER O WITH
DIAERESIS' (U+00F6) normalization form "D"

print urlencode(grapheme_substr( "ao" . $char_a_ring_nfd . "bc" .
```

```
$char_o_diaeresis_nfd . "O", 2, -1 ));
?>
```

The above example will output:

```
a%CC%8Abco%CC%88
```

See Also

- grapheme_extract()
- **grapheme_extractB()**
- » Unicode Text Segmentation: Grapheme Cluster Boundaries

intl_error_name

intl_error_name -- Get symbolic name for a given error code

Description

string **intl_error_name** (integer `$error_code`)

Return ICU error code name.

Parameters

`error_code`
ICU error code.

Return Values

The returned string will be the same as the name of the error code constant.

Examples

Example #12 - intl_error_name() **example**

```
<?php
$coll     = collator_create( 'en_RU' );
$err_code = collator_get_error_code( $coll );

printf( "Symbolic name for %d is %s\n.", $err_code, intl_error_name(
$err_code ) );
?>
```

The above example will output something similar to:

```
Symbolic name for -128 is U_USING_FALLBACK_WARNING.
```

See Also

- intl_is_failure()
- intl_get_error_code()
- intl_get_error_message()

intl_get_error_code

intl_get_error_code -- Get the last error code

Description

integer **intl_get_error_code** (void)

Useful to handle errors occured in static methods when there's no object to get error code from.

Return Values

Error code returned by the last API function call.

Examples

Example #13 - intl_get_error_code() **example**

```
<?php
$coll = collator_create( '<bad_param>' );
if( !$coll ) {
   handle_error( intl_get_error_code() );
}
?>
```

See Also

- intl_is_failure()
- intl_error_name()
- intl_get_error_message()
- collator_get_error_code()
- numfmt_get_error_code()

intl_get_error_message

intl_get_error_message -- Get description of the last error

Description

string **intl_get_error_message** (void)

Get error message from last internationalization function called.

Return Values

Description of an error occurred in the last API function call.

Examples

Example #14 - intl_get_error_message() **example**

```
<?php
if( Collator::getAvailableLocales() === false ) {
   show_error( intl_get_error_message() );
}
?>
```

See Also

- intl_error_name()
- intl_get_error_code()
- intl_is_failure()
- collator_get_error_message()
- numfmt_get_error_message()

intl_is_failure

intl_is_failure -- Check whether the given error code indicates failure

Description

bool **intl_is_failure** (integer `$error_code`)

Parameters

`error_code`
: is a value that returned by functions: intl_get_error_code(), collator_get_error_code().

Return Values

TRUE if it the code indicates some failure, and **FALSE** in case of success or a warning.

Examples

Example #15 - intl_is_failure() **example**

```
<?php
function check( $err_code )
{
   var_export( intl_is_failure( $err_code ) );
   echo "\n";
}

check( U_ZERO_ERROR );
check( U_USING_FALLBACK_WARNING );
check( U_ILLEGAL_ARGUMENT_ERROR );
?>
```

The above example will output something similar to:

```
false
false
true
```

See Also

- intl_get_error_code()
- collator_get_error_code()
- Collator-getErrorCode()

The Collator class

Introduction

Provides string comparison capability with support for appropriate locale-sensitive sort orderings.

Class synopsis

Collator

Collator {

/* Methods */

Collator::__construct (string `$locale`)

bool **Collator::asort** (array `&$arr` [, integer `$sort_flag`])

integer **Collator::compare** (string `$str1`, string `$str2`)

static Collator **Collator::create** (string `$locale`)

integer **Collator::getAttribute** (integer `$attr`)

integer **Collator::getErrorCode** (void)

string **Collator::getErrorMessage** (void)

string **Collator::getLocale** ([integer `$type`])

integer **Collator::getStrength** (void)

bool **Collator::setAttribute** (integer `$attr`, integer `$val`)

bool **Collator::setStrength** (integer `$strength`)

bool **Collator::sortWithSortKeys** (array `&$arr`)

bool **Collator::sort** (array `&$arr` [, integer `$sort_flag`])
}

Predefined Constants

Collator::FRENCH_COLLATION (integer)

Sort strings with different accents from the back of the string. This attribute is automatically set to *On* for the French locales and a few others. Users normally would not need to explicitly set this attribute. There is a string comparison performance cost when it is set *On*, but sort key length is unaffected. Possible values are:

- **Collator::ON**
- **Collator::OFF** (default)
- **Collator::DEFAULT_VALUE**

Example #16 - FRENCH_COLLATION rules

- F=OFF cote < coté < côte < côté
- F=ON cote < côte < coté < côté

Collator::ALTERNATE_HANDLING (integer)

The Alternate attribute is used to control the handling of the socalled variable characters in the UCA: whitespace, punctuation and symbols. If Alternate is set to *NonIgnorable* (N), then differences among these characters are of the same importance as differences among letters. If Alternate is set to *Shifted* (S), then these characters are of only minor importance. The *Shifted* value is often used in combination with *Strength* set to Quaternary. In such a case, whitespace, punctuation, and symbols are considered when comparing strings, but only if all other aspects of the strings (base letters, accents, and case) are identical. If Alternate is not set to Shifted, then there is no difference between a Strength of 3 and a Strength of 4. For more information and examples, see Variable_Weighting in the » UCA. The reason the Alternate values are not simply *On* and *Off* is that additional Alternate values may be added in the future. The UCA option Blanked is expressed with Strength set to 3, and Alternate set to Shifted. The default for most locales is NonIgnorable. If Shifted is selected, it may be slower if there are many strings that are the same except for punctuation; sort key length will not be affected unless the strength level is also increased. Possible values are:

- **Collator::NON_IGNORABLE** (default)
- **Collator::SHIFTED**
- **Collator::DEFAULT_VALUE**

Example #17 - ALTERNATE_HANDLING rules

- S=3, A=N di Silva < Di Silva < diSilva < U.S.A. < USA
- S=3, A=S di Silva = diSilva < Di Silva < U.S.A. = USA
- S=4, A=S di Silva < diSilva < Di Silva < U.S.A. < USA

Collator::CASE_FIRST (integer)

The Case_First attribute is used to control whether uppercase letters come before lowercase letters or vice versa, in the absence of other differences in the strings. The possible values are *Uppercase_First* (U) and *Lowercase_First* (L), plus the standard *Default* and *Off*. There is almost no difference between the Off and Lowercase_First options in terms of results, so typically users will not use Lowercase_First: only Off or Uppercase_First. (People interested in the detailed differences between X and L should consult the Collation Customization). Specifying either L or U won't affect string comparison performance, but will affect the sort key length. Possible values are:

- **Collator::OFF** (default)
- **Collator::LOWER_FIRST**
- **Collator::UPPER_FIRST**
- **Collator:DEFAULT**

Example #18 - CASE_FIRST rules

- C=X or C=L "china" < "China" < "denmark" < "Denmark"
- C=U "China" < "china" < "Denmark" < "denmark"

Collator::CASE_LEVEL (integer)

The Case_Level attribute is used when ignoring accents but not case. In such a situation, set Strength to be *Primary*, and Case_Level to be *On*. In most locales, this setting is Off by default. There is a small string comparison performance and sort key impact if this attribute is set to be *On*. Possible values are:

- **Collator::OFF** (default)
- **Collator::ON**
- **Collator::DEFAULT_VALUE**

Example #19 - CASE_LEVEL rules

- S=1, E=X role = Role = rôle
- S=1, E=O role = rôle < Role

Collator::NORMALIZATION_MODE (integer)

The Normalization setting determines whether text is thoroughly normalized or not in comparison. Even if the setting is off (which is the default for many locales), text as represented in common usage will compare correctly (for details, see UTN #5). Only if the accent marks are in noncanonical order will there be a problem. If the setting is *On*, then the best results are guaranteed for all possible text input. There is a medium string comparison performance cost if this attribute is *On*, depending on the frequency of sequences that require normalization. There is no significant effect on sort key length. If the input text is known to be in NFD or NFKD normalization forms, there is no need to enable this Normalization option. Possible values are:

- **Collator::OFF** (default)
- **Collator::ON**
- **Collator::DEFAULT_VALUE**

Collator::STRENGTH (integer)

The ICU Collation Service supports many levels of comparison (named "Levels", but also known as "Strengths"). Having these categories enables ICU to sort strings precisely according to local conventions. However, by allowing the levels to be selectively employed, searching for a string in text can be performed with various matching conditions. For more detailed information, see collator_set_strength() chapter. Possible values are:

- **Collator::PRIMARY**
- **Collator::SECONDARY**
- **Collator::TERTIARY** (default)
- **Collator::QUATERNARY**
- **Collator::IDENTICAL**
- **Collator::DEFAULT_VALUE**

Collator::HIRAGANA_QUATERNARY_MODE (integer)

Compatibility with JIS x 4061 requires the introduction of an additional level to distinguish Hiragana and Katakana characters. If compatibility with that standard is required, then this attribute should be set *On*, and the strength set to Quaternary. This will affect sort key length and string comparison string comparison performance. Possible values are:

- **Collator::OFF** (default)
- **Collator::ON**
- **Collator::DEFAULT_VALUE**

Collator::NUMERIC_COLLATION (integer)

When turned on, this attribute generates a collation key for the numeric value of substrings of digits. This is a way to get '100' to sort AFTER '2'. Possible values are:

- **Collator::OFF** (default)
- **Collator::ON**
- **Collator::DEFAULT_VALUE**

Collator::DEFAULT_VALUE (integer)

Collator::PRIMARY (integer)

Collator::SECONDARY (integer)

Collator::TERTIARY (integer)

Collator::DEFAULT_STRENGTH (integer)

Collator::QUATERNARY (integer)

Collator::IDENTICAL (integer)

Collator::OFF (integer)

Collator::ON (integer)

Collator::SHIFTED (integer)

Collator::NON_IGNORABLE (integer)

Collator::LOWER_FIRST (integer)

Collator::UPPER_FIRST (integer)

Collator::asort

collator_asort

Collator::asort -- collator_asort -- Sort array maintaining index association

Description

Object oriented style

bool **Collator::asort** (array `&$arr` [, integer `$sort_flag`])

Procedural style

bool **collator_asort** (Collator `$coll`, array `&$arr` [, integer `$sort_flag`])

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant. Array elements will have sort order according to current locale rules.

Equivalent to standard PHP asort().

Parameters

`coll`
Collator object.

`arr`
Array of strings to sort.

`sort_flag`
Optional sorting type, one of the following:

- **Collator::SORT_REGULAR** - compare items normally (don't change types)
- **Collator::SORT_NUMERIC** - compare items numerically
- **Collator::SORT_STRING** - compare items as strings

Default $sort_flag value is **Collator::SORT_REGULAR**. It is also used if an invalid $sort_flag value has been specified.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #20 - collator_asort() example

```
<?php
$coll = collator_create( 'en_US' );
$arr = array(
    'a' => '100',
    'b' => '50',
    'c' => '7'
);
collator_asort( $coll, $arr, Collator::SORT_NUMERIC );
var_export( $arr );

collator_asort( $coll, $arr, Collator::SORT_STRING );
var_export( $arr );
?>
```

The above example will output:

```
array (
 'c' => '7',
 'b' => '50',
 'a' => '100',
)array (
 'a' => '100',
 'b' => '50',
 'c' => '7',
)
```

See Also

- Collator constants
- collator_sort()
- collator_sort_with_sort_keys()

Collator::compare

collator_compare

Collator::compare -- collator_compare -- Compare two Unicode strings

Description

Object oriented style

integer **Collator::compare** (string `$str1`, string `$str2`)

Procedural style

integer **collator_compare** (Collator `$coll`, string `$str1`, string `$str2`)

Compare two Unicode strings according to collation rules.

Parameters

`coll`
: Collator object.

`str1`
: The first string to compare.

`str2`
: The second string to compare.

Return Values

Return comparison result:

- 1 if *str1* is *greater* than *str2*;
- 0 if *str1* is *equal* to *str2*;
- -1 if *str1* is *less* than *str2*.

On error boolean **FALSE** is returned.

Warning
This function may return Boolean **FALSE**, but may also return a non-Boolean value

which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Examples

Example #21 - collator_compare() example

```
<?php
$s1 = 'Hello';
$s2 = 'hello';

$coll = collator_create( 'en_US' );
$res  = collator_compare( $coll, $s1, $s2 );

if ($res === false) {
   echo collator_get_error_message( $coll );
} else if( $res > 0 ) {
   echo "s1 is greater than s2\n";
} else if( $res < 0 ) {
   echo "s1 is less than s2\n";
} else {
   echo "s1 is equal to s2\n";
?>
```

The above example will output:

```
    s1 is greater than s2
```

See Also

- collator_sort()

Collator::__construct

Collator::__construct -- Create a collator

Description

Collator::__construct (string `$locale`)

Creates a new instance of Collator.

Parameters

`locale`
: The locale whose collation rules should be used. Special values for locales can be passed in - if null is passed for the locale, the default locale collation rules will be used. If empty string ("") or "root" are passed, UCA rules will be used. The Locale attribute is typically the most important attribute for correct sorting and matching, according to the user expectations in different countries and regions. The default » UCA ordering will only sort a few languages such as Dutch and Portuguese correctly ("correctly" meaning according to the normal expectations for users of the languages). Otherwise, you need to supply the locale to UCA in order to properly collate text for a given language. Thus a locale needs to be supplied so as to choose a collator that is correctly tailored for that locale. The choice of a locale will automatically preset the values for all of the attributes to something that is reasonable for that locale. Thus most of the time the other attributes do not need to be explicitly set. In some cases, the choice of locale will make a difference in string comparison performance and/or sort key length.

Return Values

Returns Collator instance.

Errors/Exceptions

Returns an "empty" object on error. You can use intl_get_error_code() and/or intl_get_error_message() to know what happened.

Examples

Example #22 - Collator::__construct() example

```
<?php
$coll = new Collator( 'en_CA' );
?>
```

See Also

- Collator::create()
- collator_create()

Collator::create

collator_create

Collator::create -- collator_create -- Create a collator

Description

Object oriented style

static Collator **Collator::create** (string `$locale`)

Procedural style

Collator **collator_create** (string `$locale`)

The strings will be compared using the options already specified.

Parameters

`locale`
: The locale containing the required collation rules. Special values for locales can be passed in - if null is passed for the locale, the default locale collation rules will be used. If empty string ("") or "root" are passed, UCA rules will be used.

Return Values

Return new instance of Collator object, or **NULL** on error.

Examples

Example #23 - collator_create() example

```
<?php
$coll = collator_create( 'en_US' );

if( !isset( $coll ) ) {
   printf( "Collator creation failed: %s\n", intl_get_error_message() );
   exit( 1 );
}
?>
```

See Also

- Collator::__construct()

Collator::getAttribute

collator_get_attribute

Collator::getAttribute -- collator_get_attribute -- Get collation attribute value

Description

Object oriented style

integer **Collator::getAttribute** (integer `$attr`)

Procedural style

integer **collator_get_attribute** (Collator `$coll`, integer `$attr`)

Get a value of an integer collator attribute.

Parameters

`coll`
Collator object.

`attr`
Attribute to get value for.

Return Values

Attribute value, or boolean **FALSE** on error.

Examples

Example #24 - collator_get_attribute() example

```
<?php
$coll = collator_create( 'en_CA' );
$val = collator_get_attribute( $coll, Collator::NUMERIC_COLLATION );
if( $val === false )
{
   // Handle error.
}
?>
```

See Also

- Collator constants
- collator_set_attribute()
- collator_get_strength()

Collator::getErrorCode

collator_get_error_code

Collator::getErrorCode -- collator_get_error_code -- Get collator's last error code

Description

Object oriented style

integer **Collator::getErrorCode** (void)

Procedural style

integer **collator_get_error_code** (Collator `$coll`)

Parameters

`coll`
Collator object.

Return Values

Error code returned by the last Collator API function call.

Examples

Example #25 - collator_get_error_code() example

```
<?php
$coll = collator_create( 'en_US' );
if( collator_get_attribute( $coll, Collator::FRENCH_COLLATION ) === false )
       handle_error( collator_get_error_code() );
?>
```

See Also

- collator_get_error_message()

Collator::getErrorMessage

collator_get_error_message

Collator::getErrorMessage -- collator_get_error_message -- Get text for collator's last error code

Description

Object oriented style

string **Collator::getErrorMessage** (void)

Procedural style

string **collator_get_error_message** (Collator `$coll`)

Retrieves the message for the last error.

Parameters

`coll`
Collator object.

Return Values

Description of an error occurred in the last Collator API function call.

Examples

Example #26 - collator_get_error_message() example

```
<?php
$coll = collator_create( 'lt' );
if( collator_compare( $coll, 'y', 'k' ) === false ) {
   echo collator_get_error_message( $coll );
}
?>
```

See Also

- collator_get_error_code()

Collator::getLocale

collator_get_locale

Collator::getLocale -- collator_get_locale -- Get the locale name of the collator

Description

Object oriented style

string **Collator::getLocale** ([integer `$type`])

Procedural style

string **collator_get_locale** (Collator `$coll`, integer `$type`)

Get collector locale name.

Parameters

`coll`
Collator object.

`type`
You can choose between valid and actual locale (**Locale::VALID_LOCALE** and **Locale::ACTUAL_LOCALE**, respectively). The default is the actual locale.

Return Values

Real locale name from which the collation data comes. If the collator was instantiated from rules or an error occurred, returns boolean **FALSE**.

Examples

Example #27 - collator_get_locale() example

```
<?php
$coll    = collator_create( 'en_US_California' );
$res_val = collator_get_locale( $coll, Locale::VALID_LOCALE );
$res_act = collator_get_locale( $coll, Locale::ACTUAL_LOCALE );
printf( "Valid locale name: %s\nActual locale name: %s\n",
        $res_req, $res_val, $res_act );
?>
```

The above example will output:

```
Requested locale name: en_US_California
Valid locale name: en_US
Actual locale name: en
```

See Also

- collator_create()

Collator::getStrength

collator_get_strength

Collator::getStrength -- collator_get_strength -- Get current collation strength

Description

Object oriented style

integer **Collator::getStrength** (void)

Procedural style

integer **collator_get_strength** (Collator `$coll`)

Parameters

`coll`
Collator object.

Return Values

Returns current collation strength, or boolean **FALSE** on error.

Examples

Example #28 - collator_get_strength() example

```
<?php
$coll     = collator_create( 'en_US' );
$strength = collator_get_strength( $coll );
?>
```

The above example will output:

```
// TODO
```

See Also

- Collator constants
- collator_set_strength()
- collator_get_attribute()

Collator::setAttribute

collator_set_attribute

Collator::setAttribute -- collator_set_attribute -- Set collation attribute

Description

Object oriented style

bool **Collator::setAttribute** (integer `$attr`, integer `$val`)

Procedural style

bool **collator_set_attribute** (Collator `$coll`, integer `$attr`, integer `$val`)

Parameters

`coll`
Collator object.

`attr`
Attribute.

`val`
Attribute value.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #29 - collator_set_attribute() example

```
<?php
$coll = collator_create( 'en_CA' );
$val  = collator_get_attribute( $coll, Collator::NUMERIC_COLLATION );
if ($val === false) {
   // Handle error.
} elseif ($val === Collator::ON) {
   // Do something useful.
```

```
}
?>
```

See Also

- Collator constants
- collator_get_attribute()
- collator_set_strength()

Collator::setStrength

collator_set_strength

Collator::setStrength -- collator_set_strength -- Set collation strength

Description

Object oriented style

bool **Collator::setStrength** (integer `$strength`)

Procedural style

bool **collator_set_strength** (Collator `$coll`, integer `$strength`)

The » ICU Collation Service supports many levels of comparison (named "Levels", but also known as "Strengths"). Having these categories enables ICU to sort strings precisely according to local conventions. However, by allowing the levels to be selectively employed, searching for a string in text can be performed with various matching conditions.

- *Primary Level*: Typically, this is used to denote differences between base characters (for example, "a" < "b"). It is the strongest difference. For example, dictionaries are divided into different sections by base character. This is also called the level1 strength.
- *Secondary Level*: Accents in the characters are considered secondary differences (for example, "as" < "às" < "at"). Other differences between letters can also be considered secondary differences, depending on the language. A secondary difference is ignored when there is a primary difference anywhere in the strings. This is also called the level2 strength.

Note
Note: In some languages (such as Danish), certain accented letters are considered to be separate base characters. In most languages, however, an accented letter only has a secondary difference from the unaccented version of that letter.

- *Tertiary Level*: Upper and lower case differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò"). In addition, a variant of a letter differs from the base form on the tertiary level (such as "A" and " "). Another example is the difference between large and small Kana. A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings. This is also called the level3 strength.
- *Quaternary Level*: When punctuation is ignored (see Ignoring Punctuations) at level

13, an additional level can be used to distinguish words with and without punctuation (for example, "ab" < "a-b" < "aB"). This difference is ignored when there is a primary, secondary or tertiary difference. This is also known as the level4 strength. The quaternary level should only be used if ignoring punctuation is required or when processing Japanese text (see Hiragana processing).

- *Identical Level*: When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the NFD form of each string are compared at this level, just in case there is no difference at levels 14. For example, Hebrew cantillation marks are only distinguished at this level. This level should be used sparingly, as only code point values differences between two strings is an extremely rare occurrence. Using this level substantially decreases the performance for both incremental comparison and sort key generation (as well as increasing the sort key length). It is also known as level 5 strength.

For example, people may choose to ignore accents or ignore accents and case when searching for text. Almost all characters are distinguished by the first three levels, and in most locales the default value is thus Tertiary. However, if Alternate is set to be Shifted, then the Quaternary strength can be used to break ties among whitespace, punctuation, and symbols that would otherwise be ignored. If very fine distinctions among characters are required, then the Identical strength can be used (for example, Identical Strength distinguishes between the Mathematical Bold Small A and the Mathematical Italic Small A.). However, using levels higher than Tertiary the Identical strength result in significantly longer sort keys, and slower string comparison performance for equal strings.

Parameters

`coll`
Collator object.

`strength`
Strength to set. Possible values are:

- **Collator::PRIMARY**
- **Collator::SECONDARY**
- **Collator::TERTIARY**
- **Collator::QUATERNARY**
- **Collator::IDENTICAL**
- **Collator::DEFAULT**

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #30 - collator_set_strength() example

```
<?php
$arr  = array( 'aò', 'Ao', 'ao' );
$coll = collator_create( 'en_US' );

// Sort array using default strength.
collator_sort( $coll, $arr );
var_export( $arr );

// Sort array using primary strength.
collator_set_strength( $coll, Collator::PRIMARY );
collator_sort( $coll, $arr );
var_export( $arr );
?>
```

The above example will output:

```
array (
 0 => 'ao',
 1 => 'Ao',
 2 => 'aò',
)
array (
 0 => 'aò',
 1 => 'Ao',
 2 => 'ao',
)
```

See Also

- Collator constants
- collator_get_strength()

Collator::sortWithSortKeys

collator_sort_with_sort_keys

Collator::sortWithSortKeys -- collator_sort_with_sort_keys -- Sort array using specified collator and sort keys

Description

Object oriented style

bool **Collator::sortWithSortKeys** (array `&$arr`)

Procedural style

bool **collator_sort_with_sort_keys** (Collator `$coll`, array `&$arr`)

Similar to collator_sort() but uses ICU sorting keys produced by ucol_getSortKey() to gain more speed on large arrays.

Parameters

`coll`
Collator object.

`arr`
Array of strings to sort

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #31 - collator_sort_with_sort_keys() example

```
<?php
$arr  = array( 'Köpfe', 'Kypper', 'Kopfe' );
$coll = collator_create( 'sv' );

collator_sort_with_sort_keys( $coll, $arr );
var_export( $arr );
?>
```

The above example will output:

```
array (
 0 => 'Kopfe',
 1 => 'Kypper',
 2 => 'Köpfe',
)
```

See Also

- Collator constants
- collator_sort()
- collator_asort()

Collator::sort

collator_sort

Collator::sort -- collator_sort -- Sort array using specified collator

Description

Object oriented style

bool **Collator::sort** (array `&$arr` [, integer `$sort_flag`])

Procedural style

bool **collator_sort** (Collator `$coll`, array `&$arr` [, integer `$sort_flag`])

This function sorts an array according to current locale rules.

Equivalent to standard PHP sort().

Parameters

`coll`
Collator object.

`arr`
Array of strings to sort.

`sort_flag`
Optional sorting type, one of the following:

- **Collator::SORT_REGULAR** - compare items normally (don't change types)
- **Collator::SORT_NUMERIC** - compare items numerically
- **Collator::SORT_STRING** - compare items as strings

Default sorting type is **Collator::SORT_REGULAR**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #32 - collator_sort() example

```
<?php
$coll = collator_create( 'en_US' );
$arr  = array( 'at', 'às', 'as' );

var_export( $arr );
collator_sort( $coll, $arr );
var_export( $arr );
?>
```

The above example will output:

```
array (
 0 => 'at',
 1 => 'às',
 2 => 'as',
)array (
 0 => 'as',
 1 => 'às',
 2 => 'at',
)
```

See Also

- Collator constants
- collator_asort()
- collator_sort_with_sort_keys()

The NumberFormatter class

Introduction

Programs store and operate on numbers using a locale-independent binary representation. When displaying or printing a number it is converted to a locale-specific string. For example, the number 12345.67 is "12,345.67" in the US, "12 345,67" in France and "12.345,67" in Germany.

By invoking the methods provided by the NumberFormatter class, you can format numbers, currencies, and percentages according to the specified or default locale. NumberFormatter is locale-sensitive so you need to create a new NumberFormatter for each locale. NumberFormatter methods format primitive-type numbers, such as double and output the number as a locale-specific string.

For currencies you can use currency format type to create a formatter that returns a string with the formatted number and the appropriate currency sign. Of course, the NumberFormatter class is unaware of exchange rates so, the number output is the same regardless of the specified currency. This means that the same number has different monetary values depending on the currency locale. If the number is 9988776.65 the results will be:

- 9 988 776,65 ? in France
- 9.988.776,65 ? in Germany
- $9,988,776.65 in the United States

In order to format percentages, create a locale-specific formatter with percentage format type. With this formatter, a decimal fraction such as 0.75 is displayed as 75%.

For more complex formatting, like spelled-out numbers, the rule-based number formatters are used.

Class synopsis

NumberFormatter

NumberFormatter {

/* Methods */

NumberFormatter::__construct (string `$locale`, integer `$style` [, string `$pattern`])

static NumberFormatter **NumberFormatter::create** (string `$locale`, integer `$style` [, string `$pattern`])

string **NumberFormatter::formatCurrency** (double `$value`, string `$currency`)

string **NumberFormatter::format** (number `$value` [, integer `$type`])

integer **NumberFormatter::getAttribute** (integer `$attr`)

integer **NumberFormatter::getErrorCode** (void)

string **NumberFormatter::getErrorMessage** (void)

string **NumberFormatter::getLocale** ([integer `$type`])

string **NumberFormatter::getPattern** (void)

string **NumberFormatter::getSymbol** (integer `$attr`)

string **NumberFormatter::getTextAttribute** (integer `$attr`)

double **NumberFormatter::parseCurrency** (string `$value`, string `&$currency` [, integer `&$position`])

mixed **NumberFormatter::parse** (string `$value` [, integer `$type` [, integer `&$position`]])

bool **NumberFormatter::setAttribute** (integer `$attr`, integer `$value`)

bool **NumberFormatter::setPattern** (string `$pattern`)

bool **NumberFormatter::setSymbol** (integer `$attr`, string `$value`)

bool **NumberFormatter::setTextAttribute** (integer `$attr`, string `$value`)
}

Predefined Constants

These styles are used by the numfmt_create() to define the type of the formatter.

NumberFormatter::PATTERN_DECIMAL (integer)
Decimal format defined by pattern

NumberFormatter::DECIMAL (integer)
Decimal format

NumberFormatter::CURRENCY (integer)
Currency format

NumberFormatter::PERCENT (integer)

Percent format

NumberFormatter::SCIENTIFIC (integer)
Scientific format

NumberFormatter::SPELLOUT (integer)
Spellout rule-based format

NumberFormatter::ORDINAL (integer)
Ordinal rule-based format

NumberFormatter::DURATION (integer)
Duration rule-based format

NumberFormatter::PATTERN_RULEBASED (integer)
Rule-based format defined by pattern

NumberFormatter::DEFAULT_STYLE (integer)
Default format for the locale

NumberFormatter::IGNORE (integer)
Alias for PATTERN_DECIMAL

These constants define how the numbers are parsed or formatted. They should be used as arguments to numfmt_format() and numfmt_parse().
NumberFormatter::TYPE_DEFAULT (integer)
Derive the type from variable type

NumberFormatter::TYPE_INT32 (integer)
Format/parse as 32-bit integer

NumberFormatter::TYPE_INT64 (integer)
Format/parse as 64-bit integer

NumberFormatter::TYPE_DOUBLE (integer)
Format/parse as floating point value

NumberFormatter::TYPE_CURRENCY (integer)
Format/parse as currency value

Number format attribute used by numfmt_get_attribute() and numfmt_set_attribute().
NumberFormatter::PARSE_INT_ONLY (integer)
Parse integers only.

NumberFormatter::GROUPING_USED (integer)
Use grouping separator.

NumberFormatter::DECIMAL_ALWAYS_SHOWN (integer)

Always show decimal point.

NumberFormatter::MAX_INTEGER_DIGITS (integer)
Maximum integer digits.

NumberFormatter::MIN_INTEGER_DIGITS (integer)
Minimum integer digits.

NumberFormatter::INTEGER_DIGITS (integer)
Integer digits.

NumberFormatter::MAX_FRACTION_DIGITS (integer)
Maximum fraction digits.

NumberFormatter::MIN_FRACTION_DIGITS (integer)
Minimum fraction digits.

NumberFormatter::FRACTION_DIGITS (integer)
Fraction digits.

NumberFormatter::MULTIPLIER (integer)
Multiplier.

NumberFormatter::GROUPING_SIZE (integer)
Grouping size.

NumberFormatter::ROUNDING_MODE (integer)
Rounding Mode.

NumberFormatter::ROUNDING_INCREMENT (integer)
Rounding increment.

NumberFormatter::FORMAT_WIDTH (integer)
The width to which the output of format() is padded.

NumberFormatter::PADDING_POSITION (integer)
The position at which padding will take place. See pad position constants for possible argument values.

NumberFormatter::SECONDARY_GROUPING_SIZE (integer)
Secondary grouping size.

NumberFormatter::SIGNIFICANT_DIGITS_USED (integer)
Use significant digits.

NumberFormatter::MIN_SIGNIFICANT_DIGITS (integer)
Minimum significant digits.

NumberFormatter::MAX_SIGNIFICANT_DIGITS (integer)
Maximum significant digits.

NumberFormatter::LENIENT_PARSE (integer)

Lenient parse mode used by rule-based formats.

Number format text attribute used by numfmt_get_text_attribute() and numfmt_set_text_attribute().

NumberFormatter::POSITIVE_PREFIX (integer)
Positive prefix.

NumberFormatter::POSITIVE_SUFFIX (integer)
Positive suffix.

NumberFormatter::NEGATIVE_PREFIX (integer)
Negative prefix.

NumberFormatter::NEGATIVE_SUFFIX (integer)
Negative suffix.

NumberFormatter::PADDING_CHARACTER (integer)
The character used to pad to the format width.

NumberFormatter::CURRENCY_CODE (integer)
The ISO currency code.

NumberFormatter::DEFAULT_RULESET (integer)
The default rule set. This is only available with rule-based formatters.

NumberFormatter::PUBLIC_RULESETS (integer)
The public rule sets. This is only available with rule-based formatters. This is a read-only attribute. The public rulesets are returned as a single string, with each ruleset name delimited by ';' (semicolon).

Number format symbols used by numfmt_get_symbol() and numfmt_set_symbol().

NumberFormatter::DECIMAL_SEPARATOR_SYMBOL (integer)
The decimal separator.

NumberFormatter::GROUPING_SEPARATOR_SYMBOL (integer)
The grouping separator.

NumberFormatter::PATTERN_SEPARATOR_SYMBOL (integer)
The pattern separator.

NumberFormatter::PERCENT_SYMBOL (integer)
The percent sign.

NumberFormatter::ZERO_DIGIT_SYMBOL (integer)
Zero.

NumberFormatter::DIGIT_SYMBOL (integer)
Character representing a digit in the pattern.

NumberFormatter::MINUS_SIGN_SYMBOL (integer)
The minus sign.

NumberFormatter::PLUS_SIGN_SYMBOL (integer)
The plus sign.

NumberFormatter::CURRENCY_SYMBOL (integer)
The currency symbol.

NumberFormatter::INTL_CURRENCY_SYMBOL (integer)
The international currency symbol.

NumberFormatter::MONETARY_SEPARATOR_SYMBOL (integer)
The monetary separator.

NumberFormatter::EXPONENTIAL_SYMBOL (integer)
The exponential symbol.

NumberFormatter::PERMILL_SYMBOL (integer)
Per mill symbol.

NumberFormatter::PAD_ESCAPE_SYMBOL (integer)
Escape padding character.

NumberFormatter::INFINITY_SYMBOL (integer)
Infinity symbol.

NumberFormatter::NAN_SYMBOL (integer)
Not-a-number symbol.

NumberFormatter::SIGNIFICANT_DIGIT_SYMBOL (integer)
Significant digit symbol.

NumberFormatter::MONETARY_GROUPING_SEPARATOR_SYMBOL (integer)
The monetary grouping separator.

Rounding mode values used by numfmt_get_attribute() and numfmt_set_attribute() with **NumberFormatter::ROUNDING_MODE** attribute.
NumberFormatter::ROUND_CEILING (integer)
Rounding mode to round towards positive infinity.

NumberFormatter::ROUND_DOWN (integer)
Rounding mode to round towards zero.

NumberFormatter::ROUND_FLOOR (integer)
Rounding mode to round towards negative infinity.

NumberFormatter::ROUND_HALFDOWN (integer)
Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down.

NumberFormatter::ROUND_HALFEVEN (integer)
Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor.

NumberFormatter::ROUND_HALFUP (integer)
Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up.

NumberFormatter::ROUND_UP (integer)
Rounding mode to round away from zero.

Pad position values used by numfmt_get_attribute() and numfmt_set_attribute() with **NumberFormatter::PADDING_POSITION** attribute.
NumberFormatter::PAD_AFTER_PREFIX (integer)
Pad characters inserted after the prefix.

NumberFormatter::PAD_AFTER_SUFFIX (integer)
Pad characters inserted after the suffix.

NumberFormatter::PAD_BEFORE_PREFIX (integer)
Pad characters inserted before the prefix.

NumberFormatter::PAD_BEFORE_SUFFIX (integer)
Pad characters inserted before the suffix.

See Also

- » ICU formatting documentation
- » ICU number formatters
- » ICU decimal formatters
- » ICU rule-based number formatters

NumberFormatter::create

numfmt_create

NumberFormatter::__construct

NumberFormatter::create -- numfmt_create -- NumberFormatter::__construct -- Create a number formatter

Description

Object oriented style (method)

static NumberFormatter **NumberFormatter::create** (string `$locale`, integer `$style` [, string `$pattern`])

Procedural style

NumberFormatter **numfmt_create** (string `$locale`, integer `$style` [, string `$pattern`])

Object oriented style (constructor):

NumberFormatter::__construct (string `$locale`, integer `$style` [, string `$pattern`])

Creates a number formatter.

Parameters

`locale`
Locale in which the number would be formatted (locale name, e.g. en_CA).

`style`
Style of the formatting, one of the format style constants. If **NumberFormatter::PATTERN_DECIMAL** or **NumberFormatter::PATTERN_RULEBASED** is passed then the number format is opened using the given pattern, which must conform to the syntax described in » ICU DecimalFormat documentation or » ICU RuleBasedNumberFormat documentation, respectively.

`pattern`
Pattern string in case chosen style requires pattern.

Return Values

Returns NumberFormatter object or **FALSE** on error.

Examples

Example #33 - numfmt_create() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
$fmt = numfmt_create( 'it', NumberFormatter::SPELLOUT );
echo numfmt_format($fmt, 1142)."\n";
?>
```

Example #34 - NumberFormatter() example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo $fmt->format(1234567.891234567890000)."\n";
$fmt = new NumberFormatter( 'it', NumberFormatter::SPELLOUT );
echo $fmt->format(1142)."\n";
?>
```

The above example will output:

```
1.234.567,891
millicentoquarantadue
```

See Also

- numfmt_format()
- numfmt_parse()

NumberFormatter::formatCurrency

numfmt_format_currency

NumberFormatter::formatCurrency -- numfmt_format_currency -- Format a currency value

Description

Object oriented style

string **NumberFormatter::formatCurrency** (double `$value`, string `$currency`)

Procedural style

string **numfmt_format_currency** (NumberFormatter `$fmt`, double `$value`, string `$currency`)

Format the currency value according to the formatter rules.

Parameters

`fmt`
NumberFormatter object.

`value`
The numeric currency value.

`currency`
The 3-letter ISO 4217 currency code indicating the currency to use.

Return Values

String representing the formatted currency value.

Examples

Example #35 - numfmt_format_currency() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::CURRENCY );
echo numfmt_format_currency($fmt, 1234567.891234567890000, "EUR")."\n";
echo numfmt_format_currency($fmt, 1234567.891234567890000, "RUR")."\n";
$fmt = numfmt_create( 'ru_RU', NumberFormatter::CURRENCY );
echo numfmt_format_currency($fmt, 1234567.891234567890000, "EUR")."\n";
echo numfmt_format_currency($fmt, 1234567.891234567890000, "RUR")."\n";
?>
```

Example #36 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::CURRENCY );
echo $fmt->formatCurrency(1234567.891234567890000, "EUR")."\n";
echo $fmt->formatCurrency(1234567.891234567890000, "RUR")."\n";
$fmt = new NumberFormatter( 'ru_RU', NumberFormatter::CURRENCY );
echo $fmt->formatCurrency(1234567.891234567890000, "EUR")."\n";
echo $fmt->formatCurrency(1234567.891234567890000, "RUR")."\n";
?>
```

The above example will output:

```
1.234.567,89 ?
1.234.567,89 RUR
1 234 567,89?
1 234 567,89?.
```

See Also

- numfmt_get_error_code()
- numfmt_format()
- numfmt_parse_currency()

NumberFormatter::format

numfmt_format

NumberFormatter::format -- numfmt_format -- Format a number

Description

Object oriented style

string **NumberFormatter::format** (number `$value` [, integer `$type`])

Procedural style

string **numfmt_format** (NumberFormatter `$fmt`, number `$value` [, integer `$type`])

Format a numeric value according to the formatter rules.

Parameters

`fmt`
: NumberFormatter object.

`value`
: The value to format. Can be integer or double, other values will be converted to a numeric value.

`type`
: The formatting type to use.

Return Values

Returns the string containing formatted value, or **FALSE** on error.

Examples

Example #37 - numfmt_format() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
$data = numfmt_format($fmt, 1234567.891234567890000);
if(intl_is_failure(numfmt_format($fmt))) {
   report_error("Formatter error");
}
?>
```

Example #38 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
$fmt->format(1234567.891234567890000);
if(intl_is_failure($fmt->getErrorCode()) {
   report_error("Formatter error");
}
?>
```

The above example will output:

```
1.234.567,891
```

See Also

- numfmt_get_error_code()
- numfmt_format_currency()
- numfmt_parse()

NumberFormatter::getAttribute

numfmt_get_attribute

NumberFormatter::getAttribute -- numfmt_get_attribute -- Get an attribute

Description

Object oriented style

integer **NumberFormatter::getAttribute** (integer `$attr`)

Procedural style

integer **numfmt_get_attribute** (NumberFormatter `$fmt`, integer `$attr`)

Get a numeric attribute associated with the formatter. An example of a numeric attribute is the number of integer digits the formatter will produce.

Parameters

`fmt`
NumberFormatter object.

`attr`
Attribute specifier - one of the numeric attribute constants.

Return Values

Return attribute value on success, or **FALSE** on error.

Examples

Example #39 - numfmt_get_attribute() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Digits: ".numfmt_get_attribute($fmt,
NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
numfmt_set_attribute($fmt, NumberFormatter::MAX_FRACTION_DIGITS, 2);
echo "Digits: ".numfmt_get_attribute($fmt,
NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
?>
```

Example #40 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo "Digits:
".$fmt->getAttribute(NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
$fmt->setAttribute(NumberFormatter::MAX_FRACTION_DIGITS, 2);
echo "Digits:
".$fmt->getAttribute(NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Digits: 3
1.234.567,891
Digits: 2
1.234.567,89
```

See Also

- numfmt_get_error_code()
- numfmt_get_text_attribute()
- numfmt_set_attribute()

NumberFormatter::getErrorCode

numfmt_get_error_code

NumberFormatter::getErrorCode -- numfmt_get_error_code -- Get formatter's last error code.

Description

Object oriented style

integer **NumberFormatter::getErrorCode** (void)

Procedural style

integer **numfmt_get_error_code** (NumberFormatter `$fmt`)

Get error code from the last function performed by the formatter.

Parameters

`fmt`
NumberFormatter object.

Return Values

Returns error code from last formatter call.

Examples

Example #41 - numfmt_get_error_code() example

```
<?php
$fmt  = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
$data = numfmt_format($fmt, 1234567.891234567890000);
if(intl_is_failure(numfmt_get_error_code($fmt))) {
   report_error("Formatter error");
}
?>
```

Example #42 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
```

```
$fmt->format(1234567.891234567890000);
if(intl_is_failure($fmt->getErrorCode()) {
   report_error("Formatter error");
}
?>
```

See Also

- numfmt_get_error_message()
- intl_get_error_code()
- intl_is_failure()

NumberFormatter::getErrorMessage

numfmt_get_error_message

NumberFormatter::getErrorMessage -- numfmt_get_error_message -- Get formatter's last error message.

Description

Object oriented style

string **NumberFormatter::getErrorMessage** (void)

Procedural style

string **numfmt_get_error_message** (NumberFormatter `$fmt`)

Get error message from the last function performed by the formatter.

Parameters

`fmt`
NumberFormatter object.

Return Values

Returns error message from last formatter call.

Examples

Example #43 - numfmt_get_error_message() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
$data = numfmt_format($fmt, 1234567.891234567890000);
if(intl_is_failure(numfmt_get_error_code($fmt))) {
   report_error("Formatter error");
}
?>
```

Example #44 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
```

```
$fmt->format(1234567.891234567890000);
if(intl_is_failure($fmt->getErrorCode()) {
   report_error("Formatter error");
}
?>
```

See Also

- numfmt_get_error_code()
- intl_get_error_code()
- intl_is_failure()

NumberFormatter::getLocale

numfmt_get_locale

NumberFormatter::getLocale -- numfmt_get_locale -- Get formatter locale

Description

Object oriented style

string **NumberFormatter::getLocale** ([integer `$type`])

Procedural style

string **numfmt_get_locale** (NumberFormatter `$fmt` [, integer `$type`])

Get formatter locale name.

Parameters

`fmt`
NumberFormatter object.

`type`
You can choose between valid and actual locale (**Locale::VALID_LOCALE**, **Locale::ACTUAL_LOCALE**, respectively). The default is the actual locale.

Return Values

The locale name used to create the formatter.

Examples

Example #45 - numfmt_get_locale() example

```
<?php
$req     = 'fr_FR_PARIS';
$fmt     = numfmt_create( $req,  NumberFormatter::DECIMAL);
$res_val = numfmt_get_locale( $fmt, Locale::VALID_LOCALE );
$res_act = numfmt_get_locale( $fmt, Locale::ACTUAL_LOCALE );
printf( "Requested locale name: %s\nValid locale name: %s\nActual locale
name: %s\n",
        $req, $res_val, $res_act );
?>
```

The above example will output:

```
Requested locale name: fr_FR_PARIS
Valid locale name: fr_FR
Actual locale name: fr
```

See Also

- numfmt_create()
- numfmt_get_error_code()

NumberFormatter::getPattern

numfmt_get_pattern

NumberFormatter::getPattern -- numfmt_get_pattern -- Get formatter pattern

Description

Object oriented style

string **NumberFormatter::getPattern** (void)

Procedural style

string **numfmt_get_pattern** (NumberFormatter `$fmt`)

Extract pattern used by the formatter.

Parameters

`fmt`
NumberFormatter object.

Return Values

Pattern string that is used by the formatter, or **FALSE** if an error happens.

Examples

Example #46 - numfmt_get_pattern() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Pattern: ".numfmt_get_pattern($fmt)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
numfmt_set_pattern($fmt, "#0.# kg");
echo "Pattern: ".numfmt_get_pattern($fmt)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
?>
```

Example #47 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
```

```
echo "Pattern: ".$fmt->getPattern()."\n";
echo $fmt->format(1234567.891234567890000)."\n";
$fmt->setPattern("#0.# kg");
echo "Pattern: ".$fmt->getPattern()."\n";
echo $fmt->format(1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Pattern: #,##0.###
1.234.567,891
Pattern: #0.# kg
1234567,9 kg
```

See Also

- numfmt_get_error_code()
- numfmt_set_pattern()
- numfmt_create()

NumberFormatter::getSymbol

numfmt_get_symbol

NumberFormatter::getSymbol -- numfmt_get_symbol -- Get a symbol value

Description

Object oriented style

string **NumberFormatter::getSymbol** (integer `$attr`)

Procedural style

string **numfmt_get_symbol** (NumberFormatter `$fmt`, integer `$attr`)

Get a symbol associated with the formatter. The formatter uses symbols to represent the special locale-dependent characters in a number, for example the percent sign. This API is not supported for rule-based formatters.

Parameters

`fmt`
NumberFormatter object.

`attr`
Symbol specifier, one of the format symbol constants.

Return Values

The symbol string or **FALSE** on error.

Examples

Example #48 - numfmt_get_symbol() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Sep: ".numfmt_get_symbol($fmt,
NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
numfmt_set_symbol($fmt, NumberFormatter::GROUPING_SEPARATOR_SYMBOL, "*");
echo "Sep: ".numfmt_get_symbol($fmt,
NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
?>
```

Example #49 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo "Sep:
".$fmt->getSymbol(NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
$fmt->setSymbol(NumberFormatter::GROUPING_SEPARATOR_SYMBOL, "*");
echo "Sep:
".$fmt->getSymbol(NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Sep: .
1.234.567,891
Sep: *
1*234*567,891
```

See Also

- numfmt_get_error_code()
- numfmt_set_symbol()

NumberFormatter::getTextAttribute

numfmt_get_text_attribute

NumberFormatter::getTextAttribute -- numfmt_get_text_attribute -- Get a text attribute

Description

Object oriented style

string **NumberFormatter::getTextAttribute** (integer `$attr`)

Procedural style

string **numfmt_get_text_attribute** (NumberFormatter `$fmt`, integer `$attr`)

Get a text attribute associated with the formatter. An example of a text attribute is the suffix for positive numbers. If the formatter does not understand the attributre, **U_UNSUPPORTED_ERROR** error is produced. Rule-based formatters only understand **NumberFormatter::DEFAULT_RULESET** and **NumberFormatter::PUBLIC_RULESETS** .

Parameters

`fmt`
NumberFormatter object.

`attr`
Attribute specifier - one of the text attribute constants.

Return Values

Return attribute value on success, or **FALSE** on error.

Examples

Example #50 - numfmt_get_text_attribute() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Prefix: ".numfmt_get_text_attribute($fmt,
NumberFormatter::NEGATIVE_PREFIX)."\n";
echo numfmt_format($fmt, -1234567.891234567890000)."\n";
numfmt_set_text_attribute($fmt, NumberFormatter::NEGATIVE_PREFIX, "MINUS");
echo "Prefix: ".numfmt_get_text_attribute($fmt,
```

```
NumberFormatter::NEGATIVE_PREFIX)."\n";
echo numfmt_format($fmt, -1234567.891234567890000)."\n";
?>
```

Example #51 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo "Prefix:
".$fmt->getTextAttribute(NumberFormatter::NEGATIVE_PREFIX)."\n";
echo $fmt->format(-1234567.891234567890000)."\n";
$fmt->setTextAttribute(NumberFormatter::NEGATIVE_PREFIX, "MINUS");
echo "Prefix:
".$fmt->getTextAttribute(NumberFormatter::NEGATIVE_PREFIX)."\n";
echo $fmt->format(-1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Prefix: -
-1.234.567,891
Prefix: MINUS
MINUS1.234.567,891
```

See Also

- numfmt_get_error_code()
- numfmt_get_attribute()
- numfmt_set_text_attribute()

NumberFormatter::parseCurrency

numfmt_parse_currency

NumberFormatter::parseCurrency -- numfmt_parse_currency -- Parse a currency number

Description

Object oriented style

double **NumberFormatter::parseCurrency** (string `$value`, string `&$currency` [, integer `&$position`])

Procedural style

double **numfmt_parse_currency** (NumberFormatter `$fmt`, string `$value`, string `&$currency` [, integer `&$position`])

Parse a string into a double and a currency using the current formatter.

Parameters

`fmt`
: NumberFormatter object.

`position`
: Parameter to receive the currency name (3-letter ISO 4217 currency code).

`position`
: Offset in the string at which to begin parsing. On return, this value will hold the offset at which parsing ended.

Return Values

The parsed numeric value or **FALSE** on error.

Examples

Example #52 - numfmt_parse_currency() **example**

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::CURRENCY );
$num = "1.234.567,89 $";
echo "We have ".numfmt_parse_currency($fmt, $num, $curr)." in $curr\n";
?>
```

Example #53 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::CURRENCY );
$num = "1.234.567,89 $";
echo "We have ".$fmt->parseCurrency($num, $curr)." in $curr\n";
?>
```

The above example will output:

```
We have 1234567.89 in USD
```

See Also

- numfmt_get_error_code()
- numfmt_parse()
- numfmt_format_currency()

NumberFormatter::parse

numfmt_parse

NumberFormatter::parse -- numfmt_parse -- Parse a number

Description

Object oriented style

mixed **NumberFormatter::parse** (string `$value` [, integer `$type` [, integer `&$position`]])

Procedural style

mixed **numfmt_parse** (NumberFormatter `$fmt`, string `$value` [, integer `$type` [, integer `&$position`]])

Parse a string into a number using the current formatter rules.

Parameters

`fmt`
: NumberFormatter object.

`type`
: The formatting type to use. By default, **NumberFormatter::TYPE_DOUBLE** is used.

`position`
: Offset in the string at which to begin parsing. On return, this value will hold the offset at which parsing ended.

Return Values

The value of the parsed number or **FALSE** on error.

Examples

Example #54 - numfmt_parse() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
$num = "1.234.567,891";
echo numfmt_parse($fmt, $num)."\n";
echo numfmt_parse($fmt, $num, NumberFormatter::TYPE_INT32)."\n";
?>
```

Example #55 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
$num = "1.234.567,891";
echo $fmt->parse($num)."\n";
echo $fmt->parse($num, NumberFormatter::TYPE_INT32)."\n";
?>
```

The above example will output:

```
1234567.891
1234567
```

See Also

- numfmt_get_error_code()
- numfmt_format()
- numfmt_parse_currency()

NumberFormatter::setAttribute

numfmt_set_attribute

NumberFormatter::setAttribute -- numfmt_set_attribute -- Set an attribute

Description

Object oriented style

bool **NumberFormatter::setAttribute** (integer `$attr`, integer `$value`)

Procedural style

bool **numfmt_set_attribute** (NumberFormatter `$fmt`, integer `$attr`, integer `$value`)

Set a numeric attribute associated with the formatter. An example of a numeric attribute is the number of integer digits the formatter will produce.

Parameters

`fmt`
NumberFormatter object.

`attr`
Attribute specifier - one of the numeric attribute constants.

`value`
The attribute value.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #56 - numfmt_set_attribute() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Digits: ".numfmt_get_attribute($fmt,
NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
numfmt_set_attribute($fmt, NumberFormatter::MAX_FRACTION_DIGITS, 2);
echo "Digits: ".numfmt_get_attribute($fmt,
```

```
NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
?>
```

Example #57 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo "Digits:
".$fmt->getAttribute(NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
$fmt->setAttribute(NumberFormatter::MAX_FRACTION_DIGITS, 2);
echo "Digits:
".$fmt->getAttribute(NumberFormatter::MAX_FRACTION_DIGITS)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Digits: 3
1.234.567,891
Digits: 2
1.234.567,89
```

See Also

- numfmt_get_error_code()
- numfmt_get_attribute()
- numfmt_set_text_attribute()

NumberFormatter::setPattern

numfmt_set_pattern

NumberFormatter::setPattern -- numfmt_set_pattern -- Set formatter pattern

Description

Object oriented style

bool **NumberFormatter::setPattern** (string `$pattern`)

Procedural style

bool **numfmt_set_pattern** (NumberFormatter `$fmt`, string `$pattern`)

Set the pattern used by the formatter. Can not be used on a rule-based formatter.

Parameters

`fmt`
NumberFormatter object.

`pattern`
Pattern in syntax described in » ICU DecimalFormat documentation.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #58 - numfmt_set_pattern() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Pattern: ".numfmt_get_pattern($fmt)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
numfmt_set_pattern($fmt, "#0.# kg");
echo "Pattern: ".numfmt_get_pattern($fmt)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
?>
```

Example #59 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo "Pattern: ".$fmt->getPattern()."\n";
echo $fmt->format(1234567.891234567890000)."\n";
$fmt->setPattern("#0.# kg");
echo "Pattern: ".$fmt->getPattern()."\n";
echo $fmt->format(1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Pattern: #,##0.###
1.234.567,891
Pattern: #0.# kg
1234567,9 kg
```

See Also

- numfmt_get_error_code()
- numfmt_create()
- numfmt_get_pattern()

NumberFormatter::setSymbol

numfmt_set_symbol

NumberFormatter::setSymbol -- numfmt_set_symbol -- Set a symbol value

Description

Object oriented style

bool **NumberFormatter::setSymbol** (integer `$attr`, string `$value`)

Procedural style

bool **numfmt_set_symbol** (NumberFormatter `$fmt`, integer `$attr`, string `$value`)

Set a symbol associated with the formatter. The formatter uses symbols to represent the special locale-dependent characters in a number, for example the percent sign. This API is not supported for rule-based formatters.

Parameters

`fmt`
NumberFormatter object.

`attr`
Symbol specifier, one of the format symbol constants.

`value`
Text for the symbol.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #60 - numfmt_set_symbol() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Sep: ".numfmt_get_symbol($fmt,
NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
numfmt_set_symbol($fmt, NumberFormatter::GROUPING_SEPARATOR_SYMBOL, "*");
```

```
echo "Sep: ".numfmt_get_symbol($fmt,
NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo numfmt_format($fmt, 1234567.891234567890000)."\n";
?>
```

Example #61 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo "Sep:
".$fmt->getSymbol(NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
$fmt->setSymbol(NumberFormatter::GROUPING_SEPARATOR_SYMBOL, "*");
echo "Sep:
".$fmt->getSymbol(NumberFormatter::GROUPING_SEPARATOR_SYMBOL)."\n";
echo $fmt->format(1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Sep: .
1.234.567,891
Sep: *
1*234*567,891
```

See Also

- numfmt_get_error_code()
- numfmt_get_symbol()

NumberFormatter::setTextAttribute

numfmt_set_text_attribute

NumberFormatter::setTextAttribute -- numfmt_set_text_attribute -- Set a text attribute

Description

Object oriented style

bool **NumberFormatter::setTextAttribute** (integer `$attr`, string `$value`)

Procedural style

bool **numfmt_set_text_attribute** (NumberFormatter `$fmt`, integer `$attr`, string `$value`)

Set a text attribute associated with the formatter. An example of a text attribute is the suffix for positive numbers. If the formatter does not understand the attribute, **U_UNSUPPORTED_ERROR** error is produced. Rule-based formatters only understand **NumberFormatter::DEFAULT_RULESET** and **NumberFormatter::PUBLIC_RULESETS** .

Parameters

`fmt`
NumberFormatter object.

`attr`
Attribute specifier - one of the text attribute constants.

`value`
Text for the attribute value.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #62 - numfmt_set_text_attribute() example

```
<?php
$fmt = numfmt_create( 'de_DE', NumberFormatter::DECIMAL );
echo "Prefix: ".numfmt_get_text_attribute($fmt,
NumberFormatter::NEGATIVE_PREFIX)."\n";
```

```
echo numfmt_format($fmt, -1234567.891234567890000)."\n";
numfmt_set_text_attribute($fmt, NumberFormatter::NEGATIVE_PREFIX, "MINUS");
echo "Prefix: ".numfmt_get_text_attribute($fmt,
NumberFormatter::NEGATIVE_PREFIX)."\n";
echo numfmt_format($fmt, -1234567.891234567890000)."\n";
?>
```

Example #63 - OO example

```
<?php
$fmt = new NumberFormatter( 'de_DE', NumberFormatter::DECIMAL );
echo "Prefix:
".$fmt->getTextAttribute(NumberFormatter::NEGATIVE_PREFIX)."\n";
echo $fmt->format(-1234567.891234567890000)."\n";
$fmt->setTextAttribute(NumberFormatter::NEGATIVE_PREFIX, "MINUS");
echo "Prefix:
".$fmt->getTextAttribute(NumberFormatter::NEGATIVE_PREFIX)."\n";
echo $fmt->format(-1234567.891234567890000)."\n";
?>
```

The above example will output:

```
Prefix: -
-1.234.567,891
Prefix: MINUS
MINUS1.234.567,891
```

See Also

- numfmt_get_error_code()
- numfmt_get_text_attribute()
- numfmt_set_attribute()

The Locale class

Introduction

A "Locale" is an identifier used to get language, culture, or regionally-specific behavior from an API. PHP locales are organized and identified the same way that the CLDR locales used by ICU (and many vendors of Unix-like operating systems, the Mac, Java, and so forth) use. Locales are identified using RFC 4646 language tags (which use hyphen, not underscore) in addition to the more traditional underscore-using identifiers. Unless otherwise noted the functions in this class are tolerant of both formats.

Examples of identifiers include:

- en-US (English, United States)
- zh-Hant-TW (Chinese, Traditional Script, Taiwan)
- fr-CA, fr-FR (French for Canada and France respectively)

The Locale class (and related procedural functions) are used to interact with locale identifiers--to verify that an ID is well-formed, valid, etc. The extensions used by CLDR in UAX #35 (and inherited by ICU) are valid and used wherever they would be in ICU normally.

Locales cannot be instantiated as objects. All of the functions/methods provided are static.

The null or empty string obtains the "root" locale. The "root" locale is equivalent to "en_US_POSIX" in CLDR. Language tags (and thus locale identifiers) are case insensitive. There exists a canonicalization function to make case match the specification.

Class synopsis

Locale

Locale {

/* Methods */

static string **Locale::composeLocale** (array `$subtags`)

static boolean **Locale::filterMatches** (string `$langtag`, string `$locale`)

static array **Locale::getAllVariants** (string `$locale`)

static string **Locale::getDefault** (void)

static string **Locale::getDisplayLanguage** (string `$locale` [, string `$in_locale`])

static string **Locale::getDisplayName** (string `$locale` [, string `$in_locale`])

static string **Locale::getDisplayRegion** (string `$locale` [, string `$in_locale`])

static string **Locale::getDisplayScript** (string `$locale` [, string `$in_locale`])

static string **Locale::getDisplayVariant** (string `$locale` [, string `$in_locale`])

static array **Locale::getKeywords** (string `$locale`)

static string **Locale::getPrimaryLanguage** (string `$locale`)

static string **Locale::getRegion** (string `$locale`)

static string **Locale::getScript** (string `$locale`)

static string **Locale::lookup** (array `$langtag`, string `$locale`, string `$default`)

static array **Locale::parseLocale** (string `$locale`)

static boolean **Locale::setDefault** (string `$locale`)
}

Predefined Constants

These constants define how the Locale
Locale::DEFAULT_LOCALE (string)
Used with the getLocale methods of the various locale affected classes, such as numfmt.

Locale::ACTUAL_LOCALE (string)
This is locale the data actually comes from.

Locale::VALID_LOCALE (string)
This is the most specific locale supported by ICU.

These constants define how the Locales are parsed or composed. They should be used as keys in the argument array to **locale_compose()** and are returned from **locale_parse()** as keys of the returned associative array.
Locale::LANG_TAG (string)
Language subtag

Locale::EXTLANG_TAG (string)
Extended language subtag

Locale::SCRIPT_TAG (string)
Script subtag

Locale::REGION_TAG (string)
Region subtag

Locale::VARIANT_TAG (string)
Variant subtag

Locale::GRANDFATHERED_LANG_TAG (string)
Grandfathered Language subtag

Locale::PRIVATE_TAG (string)
Private subtag

See Also

- » RFC 4646 - Tags for Identifying Languages
- » RFC 4647 - Matching of Language Tags
- » Unicode CLDR Project:Common Locale Data Repository
- » IANA Language Subtags Registry
- » ICU User Guide - Locale
- » ICU Locale api

Locale::composeLocale

locale_compose_locale

Locale::composeLocale -- locale_compose_locale -- Returns a correctly ordered and delimited locale ID

Description

Object oriented style

static string **Locale::composeLocale** (array `$subtags`)

Procedural style

string **locale_compose_locale** (array `$subtags`)

Returns a correctly ordered and delimited locale ID the keys identify the particular locale ID subtags, and the values are the associated subtag values.

Parameters

`subtags`
: an array containing a list of key-value pairs, where the keys identify the particular locale ID subtags, and the values are the associated subtag values.

Note
The 'variant' and 'private' subtags can take maximum 15 values whereas 'extlang' can take maximum 3 values.e.g. Variants are allowed with the suffix ranging from 0-14 Hence the keys for the input array can be variant0, variant1, ...,variant14 .In the returned locale id, the subtag is ordered by suffix resulting in variant0 followed by variant1 followed by variant2 and so on.

Return Values

The corresponding locale identifier.

Examples

Example #64 - locale_compose_locale() example

```
<?php
$arr = array(
   'language'=>'en' ,
   'script'  =>'Hans' ,
   'region'  =>'CN',
   'variant2'=>'rozaj' ,
   'variant1'=>'nedis' ,
   'private1'=>'prv1' ,
   'private2'=>'prv2'
);
echo locale_compose( $arr );
?>
```

Example #65 - OO example

```
<?php
$arr = array(
   'language'=>'en' ,
   'script'  =>'Hans' ,
   'region'  =>'CN',
   'variant2'=>'rozaj' ,
   'variant1'=>'nedis' ,
   'private1'=>'prv1' ,
   'private2'=>'prv2'
);
echo Locale::composeLocale( $arr );
?>
```

The above example will output:

```
Locale: en_Hans_CN_nedis_rozaj_x_prv1_prv2
```

See Also

- **locale_parse()**

Locale::filterMatches

locale_filter_matches

Locale::filterMatches -- locale_filter_matches -- Checks if a $langtag filter matches with $locale according to

Description

Object oriented style

static boolean **Locale::filterMatches** (string `$langtag`, string `$locale`)

Procedural style

boolean **locale_filter_matches** (string `$langtag`, string `$locale`)

Checks if a $langtag filter matches with $locale according to RFC 4647's basic filtering algorithm

Parameters

`langtag`
The language tag to check

`locale`
The language range to check against

Return Values

TRUE if $locale matches $langtag **FALSE** otherwise.

Examples

Example #66 - locale_filter_matches() **example**

```
<?php
echo (locale_filter_matches('de-DEVA','de-DE', false)) ? "Matches" : "Does
not match";
echo '; ';
echo (locale_filter_matches('de-DE_1996','de-DE', false)) ? "Matches" :
"Does not match";
?>
```

Example #67 - OO example

```
<?php
echo (Locale::filter_matches('de-DEVA','de-DE', false)) ? "Matches" : "Does
not match";
echo '; ';
echo (Locale::filter_matches('de-DE-1996','de-DE', false)) ? "Matches" :
"Does not match";
?>
```

The above example will output:

```
Does not match; Matches
```

See Also

- locale_lookup()

Locale::getAllVariants

locale_get_all_variants

Locale::getAllVariants -- locale_get_all_variants -- Gets the variants for the input locale

Description

Object oriented style

static array **Locale::getAllVariants** (string `$locale`)

Procedural style

array **locale_get_all_variants** (string `$locale`)

Gets the variants for the input locale

Parameters

`locale`
: The locale to extract the variants from

Return Values

The array containing the list of all variants subtag for the locale or **NULL** if not present

Examples

Example #68 - locale_get_all_variants() example

```
<?php
$arr = locale_get_all_variants('sl_IT_NEDIS_ROJAZ_1901');
var_export( $arr );
?>
```

Example #69 - OO example

```
<?php
$arr = Locale::getAllVariants('sl_IT_NEDIS_ROJAZ_1901');
var_export( $arr );
?>
```

The above example will output:

```
array (
   0 => 'NEDIS',
   1 => 'ROJAZ',
   2 => '1901',
)
```

See Also

- locale_get_primary_language()
- locale_get_script()
- locale_get_region()

Locale::getDefault

locale_get_default

Locale::getDefault -- locale_get_default -- Gets the default locale value from the INTL global 'default_locale'

Description

Object oriented style

static string **Locale::getDefault** (void)

Procedural style

string **locale_get_default** (void)

Gets the default locale value. At the PHP initialization this value is set to 'intl.default_locale' value from *php.ini* if that value exists or from ICU's function uloc_getDefault().

Parameters

Return Values

The current runtime locale

Examples

Example #70 - locale_get_default() example

```
<?php
ini_set('intl.default_locale', 'de-DE');
echo locale_get_default();
echo '; ';
locale_set_default('fr');
echo locale_get_default();
?>
```

Example #71 - OO example

```
<?php
ini_set('intl.default_locale', 'de-DE');
echo Locale::getDefault();
```

```
echo '; ';
Locale::setDefault('fr');
echo Locale::getDefault();
?>
```

The above example will output:

```
de-DE; fr
```

See Also

- locale_set_default()

Locale::getDisplayLanguage

locale_get_display_language

Locale::getDisplayLanguage -- locale_get_display_language -- Returns an appropriately localized display name for language of the inputlocale

Description

Object oriented style

static string **Locale::getDisplayLanguage** (string `$locale` [, string `$in_locale`])

Procedural style

string **locale_get_display_language** (string `$locale` [, string `$in_locale`])

Returns an appropriately localized display name for language of the input locale. If is **NULL** then the default locale is used.

Parameters

`locale`
: The locale to return a display language for

`in_locale`
: Optional format locale to use to display the language name

Return Values

display name of the language for the $locale in the format appropriate for $in_locale.

Examples

Example #72 - locale_get_display_language() **example**

```
<?php
echo locale_get_display_language('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo locale_get_display_language('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo locale_get_display_language('sl-Latn-IT-nedis', 'de');
?>
```

Example #73 - OO example

```
<?php
echo Locale::getDisplayLanguage('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo Locale::getDisplayLanguage('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo Locale::getDisplayLanguage('sl-Latn-IT-nedis', 'de');
?>
```

The above example will output:

```
Slovenian;
slov\xc3\xa8ne;
Slowenisch
```

See Also

- locale_get_display_name()
- locale_get_display_script()
- locale_get_display_region()
- locale_get_display_variant()

Locale::getDisplayName

locale_get_display_name

Locale::getDisplayName -- locale_get_display_name -- Returns an appropriately localized display name for the input locale

Description

Object oriented style

static string **Locale::getDisplayName** (string `$locale` [, string `$in_locale`])

Procedural style

string **locale_get_display_name** (string `$locale` [, string `$in_locale`])

Returns an appropriately localized display name for the input locale. If is **NULL** then the default locale is used.

Parameters

`locale`
: The locale to return a display name for.

`in_locale`
: optional format locale

Return Values

Display name of the locale in the format appropriate for $in_locale.

Examples

Example #74 - locale_get_display_name() **example**

```
<?php
echo locale_get_display_name('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo locale_get_display_name('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo locale_get_display_name('sl-Latn-IT-nedis', 'de');
?>
```

Example #75 - OO example

```
<?php
echo Locale::getDisplayName('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo Locale::getDisplayName('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo Locale::getDisplayName('sl-Latn-IT-nedis', 'de');
?>
```

The above example will output:

```
Slovenian (Latin, Italy, Natisone dialect);
slov\xc3\xa8ne (latin, Italie, dialecte de Natisone;
Slowenisch (Lateinisch, Italien, NEDIS)
```

See Also

- locale_get_display_language()
- locale_get_display_script()
- locale_get_display_region()
- locale_get_display_variant()

Locale::getDisplayRegion

locale_get_display_region

Locale::getDisplayRegion -- locale_get_display_region -- Returns an appropriately localized display name for region of the input locale

Description

Object oriented style

static string **Locale::getDisplayRegion** (string `$locale` [, string `$in_locale`])

Procedural style

string **locale_get_display_region** (string `$locale` [, string `$in_locale`])

Returns an appropriately localized display name for region of the input locale. If is **NULL** then the default locale is used.

Parameters

`locale`
: The locale to return a display region for.

`in_locale`
: Optional format locale to use to display the region name

Return Values

display name of the region for the $locale in the format appropriate for $in_locale.

Examples

Example #76 - locale_get_display_region() **example**

```
<?php
echo locale_get_display_region('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo locale_get_display_region('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo locale_get_display_region('sl-Latn-IT-nedis', 'de');
?>
```

Example #77 - OO example

```
<?php
echo Locale::getDisplayRegion('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo Locale::getDisplayRegion('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo Locale::getDisplayRegion('sl-Latn-IT-nedis', 'de');
?>
```

The above example will output:

```
Italy;
Italie;
Italien
```

See Also

- locale_get_display_name()
- locale_get_display_language()
- locale_get_display_script()
- locale_get_display_variant()

Locale::getDisplayScript

locale_get_display_script

Locale::getDisplayScript -- locale_get_display_script -- Returns an appropriately localized display name for script of the input locale

Description

Object oriented style

static string **Locale::getDisplayScript** (string `$locale` [, string `$in_locale`])

Procedural style

string **locale_get_display_script** (string `$locale` [, string `$in_locale`])

Returns an appropriately localized display name for script of the input locale. If is **NULL** then the default locale is used.

Parameters

`locale`
: The locale to return a display script for

`in_locale`
: Optional format locale to use to display the script name

Return Values

Display name of the script for the $locale in the format appropriate for $in_locale.

Examples

Example #78 - locale_get_display_script() **example**

```
<?php
echo locale_get_display_script('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo locale_get_display_script('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo locale_get_display_script('sl-Latn-IT-nedis', 'de');
?>
```

Example #79 - OO example

```
<?php
echo Locale::getDisplayScript('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo Locale::getDisplayScript('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo Locale::getDisplayScript('sl-Latn-IT-nedis', 'de');
?>
```

The above example will output:

```
Latin;
latin;
Lateinisch
```

See Also

- locale_get_display_name()
- locale_get_display_language()
- locale_get_display_region()
- locale_get_display_variant()

Locale::getDisplayVariant

locale_get_display_variant

Locale::getDisplayVariant -- locale_get_display_variant -- Returns an appropriately localized display name for variants of the input locale

Description

Object oriented style

static string **Locale::getDisplayVariant** (string `$locale` [, string `$in_locale`])

Procedural style

string **locale_get_display_variant** (string `$locale` [, string `$in_locale`])

Returns an appropriately localized display name for variants of the input locale. If is **NULL** then the default locale is used.

Parameters

`locale`
: The locale to return a display variant for

`in_locale`
: Optional format locale to use to display the variant name

Return Values

Display name of the variant for the $locale in the format appropriate for $in_locale.

Examples

Example #80 - locale_get_display_variant() **example**

```
<?php
echo locale_get_display_variant('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo locale_get_display_variant('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo locale_get_display_variant('sl-Latn-IT-nedis', 'de');
?>
```

Example #81 - OO example

```
<?php
echo Locale::getDisplayVariant('sl-Latn-IT-nedis', 'en');
echo ";\n";
echo Locale::getDisplayVariant('sl-Latn-IT-nedis', 'fr');
echo ";\n";
echo Locale::getDisplayVariant('sl-Latn-IT-nedis', 'de');
?>
```

The above example will output:

```
Natisone dialect;
dialecte de Natisone;
NEDIS
```

See Also

- locale_get_display_name()
- locale_get_display_language()
- locale_get_display_script()
- locale_get_display_region()

Locale::getKeywords

locale_get_keywords

Locale::getKeywords -- locale_get_keywords -- Gets the keywords for the input locale

Description

Object oriented style

static array **Locale::getKeywords** (string `$locale`)

Procedural style

array **locale_get_keywords** (string `$locale`)

Gets the keywords for the input locale.

Parameters

`locale`
The locale to extract the keywords from

Return Values

Associative array containing the keyword-value pairs for this locale

Examples

Example #82 - locale_get_keywords() example

```
<?php
$keywords_arr = locale_get_keywords(
'de_DE@currency=EUR;collation=PHONEBOOK' );
if ($keywords_arr) {
   foreach ($keywords_arr as $key => $value) {
       echo "$key = $value\n";
   }
}
?>
```

Example #83 - OO example

```
<?php
```

```
$keywords_arr = Locale::getKeywords(
'de_DE@currency=EUR;collation=PHONEBOOK' );
if ($keywords_arr) {
   foreach( $keywords_arr as $key => $value){
       echo "$key = $value\n";
   }
}
?>
```

The above example will output:

```
collation = PHONEBOOK
currency = EUR
```

See Also

- locale_get_all_variants()

Locale::getPrimaryLanguage

locale_get_primary_language

Locale::getPrimaryLanguage -- locale_get_primary_language -- Gets the primary language for the input locale

Description

Object oriented style

static string **Locale::getPrimaryLanguage** (string `$locale`)

Procedural style

string **locale_get_primary_language** (string `$locale`)

Gets the primary language for the input locale

Parameters

`locale`
The locale to extract the primary language code from

Return Values

The language code associated with the language or **NULL** in case of error.

Examples

Example #84 - locale_get_primary_language() **example**

```
<?php
echo locale_get_primary_language('zh-Hant');
?>
```

Example #85 - OO example

```
<?php
echo Locale::getPrimaryLanguage('zh-Hant');
?>
```

The above example will output:

```
zh
```

See Also

- locale_get_script()
- locale_get_region()
- locale_get_all_variants()

Locale::getRegion

locale_get_region

Locale::getRegion -- locale_get_region -- Gets the region for the input locale

Description

Object oriented style

static string **Locale::getRegion** (string `$locale`)

Procedural style

string **locale_get_region** (string `$locale`)

Gets the region for the input locale.

Parameters

`locale`
The locale to extract the region code from

Return Values

The region subtag for the locale or **NULL** if not present

Examples

Example #86 - locale_get_region() example

```
<?php
echo locale_get_region('de-CH-1901');
?>
```

Example #87 - OO example

```
<?php
echo Locale::getRegion('de-CH-1901');
?>
```

The above example will output:

CH

See Also

- locale_get_primary_language()
- locale_get_script()
- locale_get_all_variants()

Locale::getScript

locale_get_script

Locale::getScript -- locale_get_script -- Gets the script for the input locale

Description

Object oriented style

static string **Locale::getScript** (string `$locale`)

Procedural style

string **locale_get_script** (string `$locale`)

Gets the script for the input locale.

Parameters

`locale`
: The locale to extract the script code from

Return Values

The script subtag for the locale or **NULL** if not present

Examples

Example #88 - locale_get_script() example

```
<?php
echo locale_get_script('sr-Cyrl');
?>
```

Example #89 - OO example

```
<?php
echo Locale::getScript('sr-Cyrl');
?>
```

The above example will output:

Cyrl

See Also

- locale_get_primary_language()
- locale_get_region()
- locale_get_all_variants()

Locale::lookup

locale_lookup

Locale::lookup -- locale_lookup -- Searches the language tag list for the best match to the language

Description

Object oriented style

static string **Locale::lookup** (array `$langtag`, string `$locale`, string `$default`)

Procedural style

string **locale_lookup** (array `$langtag`, string `$locale`, string `$default`)

Searches the items in *`langtag`* for the best match to the language range specified in *`locale`* according to RFC 4647's lookup algorithm.

Parameters

`langtag`
: An array containing a list of language tags to compare to *`locale`*. Maximum 100 items allowed.

`locale`
: The locale to use as the language range when matching.

`default`
: The locale to use if no match is found.

Return Values

The closest matching language tag or default value.

Examples

Example #90 - locale_lookup() **example**

```
<?php
$arr = array(
   'de-DEVA',
   'de-DE-1996',
   'de',
```

```
   'de-De'
);
echo locale_lookup($arr, 'de-DE-1996-x-prv1-prv2', 'en_US');
?>
```

Example #91 - OO example

```
<?php
$arr = array(
   'de-DEVA',
   'de-DE-1996',
   'de',
   'de-De'
);
echo Locale::lookup($arr, 'de-DE-1996-x-prv1-prv2', 'en_US');
?>
```

The above example will output:

```
de_de_1996
```

See Also

- locale_filter_matches()

Locale::parseLocale

locale_parse_locale

Locale::parseLocale -- locale_parse_locale -- Returns a key-value array of locale ID subtag elements.

Description

Object oriented style

static array **Locale::parseLocale** (string `$locale`)

Procedural style

array **locale_parse_locale** (string `$locale`)

Returns a key-value array of locale ID subtag elements.

Parameters

`locale`
: The locale to extract the subtag array from. Note: The 'variant' and 'private' subtags can take maximum 15 values whereas 'extlang' can take maximum 3 values.

Return Values

Returns an array containing a list of key-value pairs, where the keys identify the particular locale ID subtags, and the values are the associated subtag values. The array will be ordered as the locale id subtags e.g. in the locale id if variants are '-varX-varY-varZ' then the returned array will have variant0=>varX , variant1=>varY , variant2=>varZ

Examples

Example #92 - locale_parse_locale() **example**

```
<?php
$arr = locale_parse('sl-Latn-IT-nedis');
if ($arr) {
   foreach ($arr as $key => $value) {
       echo "$key : $value , ";
   }
}
?>
```

Example #93 - OO example

```
<?php
$arr = Locale::parseLocale('sl-Latn-IT-nedis');
if ($arr) {
   foreach ($arr as $key => $value) {
       echo "$key : $value , ";
   }
}
?>
```

The above example will output:

```
language : sl , script : Latn , region : IT , variant0 : NEDIS ,
```

See Also

- **compose_locale()**

Locale::setDefault

locale_set_default

Locale::setDefault -- locale_set_default -- sets the default runtime locale

Description

Object oriented style

static boolean **Locale::setDefault** (string `$locale`)

Procedural style

boolean **locale_set_default** (string `$locale`)

Sets the default runtime locale to $locale. This changes the value of INTL global 'default_locale' locale identifier. UAX #35 extensions are accepted.

Parameters

`locale`
Is a BCP 47 compliant language tag containing the

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #94 - locale_set_default() **example**

```
<?php
locale_set_default('de-DE');
echo locale_get_default();
?>
```

Example #95 - OO example

```
<?php
Locale::setDefault('de-DE');
echo Locale::getDefault();
?>
```

The above example will output:

```
de-DE
```

See Also

- locale_get_default()

The Normalizer class

Introduction

Normalization is a process that involves transforming characters and sequences of characters into a formally-defined underlying representation. This process is most important when text needs to be compared for sorting and searching, but it is also used when storing text to ensure that the text is stored in a consistent representation.

The Unicode Consortium has defined a number of normalization forms reflecting the various needs of applications:

- Normalization Form D (NFD) - Canonical Decomposition
- Normalization Form C (NFC) - Canonical Decomposition followed by Canonical Composition
- Normalization Form KD (NFKD) - Compatibility Decomposition
- Normalization Form KC (NFKC) - Compatibility Decomposition followed by Canonical Composition

The different forms are defined in terms of a set of transformations on the text, transformations that are expressed by both an algorithm and a set of data files.

Class synopsis

Normalizer

Normalizer {

/* Methods */

static boolean **Normalizer::isNormalized** (string `$input` [, string `$form`])

static string **Normalizer::normalize** (string `$input` [, string `$form`])
}

Predefined Constants

The following constants define the normalization form used by the normalizer:

Normalizer::FORM_C (string)
Normalization Form C (NFC) - Canonical Decomposition followed by Canonical Composition

Normalizer::FORM_D (string)

Normalization Form D (NFD) - Canonical Decomposition

Normalizer::FORM_KC (string)
Normalization Form KC (NFKC) - Compatibility Decomposition, followed by Canonical Composition

Normalizer::FORM_KD (string)
Normalization Form KD (NFKD) - Compatibility Decomposition

Normalizer::NONE (string)
No decomposition/composition

Normalizer::OPTION_DEFAULT (string)
Default normalization options

See Also

- » Unicode Normalization
- » Unicode Normalization FAQ
- » ICU User Guide - Normalization
- » ICU API Reference - Normalization

Normalizer::isNormalized

normalizer_is_normalized

Normalizer::isNormalized -- normalizer_is_normalized -- Checks if the provided string is already in the specified normalization form.

Description

Object oriented style

static boolean **Normalizer::isNormalized** (string `$input` [, string `$form`])

Procedural style

boolean **normalizer_is_normalized** (string `$input` [, string `$form`])

Checks if the provided string is already in the specified normalization form.

Parameters

`input`
The input string to normalize

`form`
One of the normalization forms. Defaults to Normalizer::FORM_C.

Return Values

TRUE if normalized, **FALSE** otherwise or if there an error

Examples

Example #96 - normalizer_is_normalized() **example**

```
<?php
$char_A_ring = "\xC3\x85"; // 'LATIN CAPITAL LETTER A WITH RING ABOVE'
(U+00C5)
$char_combining_ring_above = "\xCC\x8A";  // 'COMBINING RING ABOVE' (U+030A)

$char_orig = 'A' . $char_combining_ring_above;
$char_norm = normalizer_normalize( 'A' . $char_combining_ring_above,
Normalizer::FORM_C );

echo ( normalizer_is_normalized($char_orig, Normalizer::FORM_C) ) ?
"normalized" : "not normalized";
```

```
echo '; ';
echo ( normalizer_is_normalized($char_norm, Normalizer::FORM_C) ) ?
"normalized" : "not normalized";
?>
```

Example #97 - OO example

```
<?php
$char_A_ring = "\xC3\x85"; // 'LATIN CAPITAL LETTER A WITH RING ABOVE'
(U+00C5)
$char_combining_ring_above = "\xCC\x8A";  // 'COMBINING RING ABOVE' (U+030A)

$char_orig = 'A' . $char_combining_ring_above;
$char_norm = Normalizer::normalize( 'A' . $char_combining_ring_above,
Normalizer::FORM_C );

echo ( Normalizer::isNormalized($char_orig, Normalizer::FORM_C) ) ?
"normalized" : "not normalized";
echo '; ';
echo ( Normalizer::isNormalized($char_norm, Normalizer::FORM_C) ) ?
"normalized" : "not normalized";
?>
```

The above example will output:

```
not normalized; normalized
```

See Also

- normalizer_normalize()

Normalizer::normalize

normalizer_normalize

Normalizer::normalize -- normalizer_normalize -- Normalizes the input provided and returns the normalized string

Description

Object oriented style

static string **Normalizer::normalize** (string `$input` [, string `$form`])

Procedural style

string **normalizer_normalize** (string `$input` [, string `$form`])

Normalizes the input provided and returns the normalized string

Parameters

`input`
The input string to normalize

`form`
One of the normalization forms. If not provided the default is Normalizer::FORM_C.

Return Values

The normalized string or **NULL** if an error occurred.

Examples

Example #98 - normalizer_normalize() **example**

```
<?php
$char_A_ring = "\xC3\x85"; // 'LATIN CAPITAL LETTER A WITH RING ABOVE'
(U+00C5)
$char_combining_ring_above = "\xCC\x8A";  // 'COMBINING RING ABOVE' (U+030A)

$char_1 = normalizer_normalize( $char_A_ring, Normalizer::FORM_C );
$char_2 = normalizer_normalize( 'A' . $char_combining_ring_above,
Normalizer::FORM_C );

echo urlencode($char_1);
echo ' ';
```

```
echo urlencode($char_2);
?>
```

Example #99 - OO example

```
<?php
$char_A_ring = "\xC3\x85"; // 'LATIN CAPITAL LETTER A WITH RING ABOVE'
(U+00C5)
$char_combining_ring_above = "\xCC\x8A";  // 'COMBINING RING ABOVE' (U+030A)

$char_1 = Normalizer::normalize( $char_A_ring, Normalizer::FORM_C );
$char_2 = Normalizer::normalize( 'A' . $char_combining_ring_above,
Normalizer::FORM_C );

echo urlencode($char_1);
echo ' ';
echo urlencode($char_2);
?>
```

The above example will output:

```
%C3%85 %C3%85
```

See Also

- normalizer_is_normalized()

The MessageFormatter class

Introduction

MessageFormatter is a concrete class that enables users to produce concatenated, language-neutral messages. The methods supplied in this class are used to build all the messages that are seen by end users.

The MessageFormatter class assembles messages from various fragments (such as text fragments, numbers, and dates) supplied by the program. Because of the MessageFormatter class, the program does not need to know the order of the fragments. The class uses the formatting specifications for the fragments to assemble them into a message that is contained in a single string within a resource bundle. For example, MessageFormatter enables you to print the phrase "Finished printing x out of y files..." in a manner that still allows for flexibility in translation.

Previously, an end user message was created as a sentence and handled as a string. This procedure created problems for localizers because the sentence structure, word order, number format and so on are very different from language to language. The language-neutral way to create messages keeps each part of the message separate and provides keys to the data. Using these keys, the MessageFormatter class can concatenate the parts of the message, localize them, and display a well-formed string to the end user.

MessageFormatter takes a set of objects, formats them, and then inserts the formatted strings into the pattern at the appropriate places. Choice formats can be used in conjunction with MessageFormatter to handle plurals, match numbers, and select from an array of items. Typically, the message format will come from resources and the arguments will be dynamically set at runtime.

Class synopsis

MessageFormatter

MessageFormatter {

/* Methods */

MessageFormatter::__construct (string `$locale`, string `$pattern`)

MessageFormatter **MessageFormatter::create** (string `$locale`, string `$pattern`)

static string **MessageFormatter::formatMessage** (string `$locale`, string `$pattern`, array `$args`)

    string **MessageFormatter::format** (array `$args`)

    integer **MessageFormatter::getErrorCode** (void)

    string **MessageFormatter::getErrorMessage** (void)

    string **MessageFormatter::getLocale** (void)

    string **MessageFormatter::getPattern** (void)

    static array **MessageFormatter::parseMessage** (string `$locale`, string `$value`)

    array **MessageFormatter::parse** (string `$value`)

    boolean **MessageFormatter::setPattern** (string `$pattern`)
}

See Also

- » ICU formatting documentation
- » ICU message formatting description
- » ICU message formatters
- » ICU choice formatters

MessageFormatter::create

MessageFormatter::__construct

msgfmt_create

MessageFormatter::create -- MessageFormatter::__construct -- msgfmt_create -- Constructs a new Message Formatter

Description

Object oriented style (method)

MessageFormatter **MessageFormatter::create** (string `$locale`, string `$pattern`)

Object oriented style (constructor):

MessageFormatter::__construct (string `$locale`, string `$pattern`)

Procedural style

MessageFormatter **msgfmt_create** (string `$locale`, string `$pattern`)

Constructs a new Message Formatter

Parameters

`locale`
The locale to use when formatting arguments

`pattern`
The pattern string to stick arguments into. The pattern uses an 'apostrophe-friendly' syntax; it is run through » umsg_autoQuoteApostrophe before being interpreted.

Return Values

The formatter object

Examples

Example #100 - msgfmt_create() example

```
<?php
```

```
$fmt = msgfmt_create("en_US", "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree");
echo msgfmt_format($fmt, array(4560, 123, 4560/123));
$fmt = msgfmt_create("de", "{0,number,integer} Affen über {1,number,integer}
Bäume um {2,number} Affen pro Baum");
echo msgfmt_format($fmt, array(4560, 123, 4560/123));
?>
```

Example #101 - OO example

```
<?php
$fmt = new MessageFormatter("en_US", "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree");
echo $fmt->format(array(4560, 123, 4560/123));
$fmt = new MessageFormatter("de", "{0,number,integer} Affen über
{1,number,integer} Bäume um {2,number} Affen pro Baum");
echo $fmt->format(array(4560, 123, 4560/123));
?>
```

The above example will output:

```
4,560 monkeys on 123 trees make 37.073 monkeys per tree
4.560 Affen über 123 Bäume um 37,073 Affen pro Baum
```

See Also

- msgfmt_format()
- msgfmt_parse()
- msgfmt_get_error_code()
- msgfmt_get_error_message()

MessageFormatter::formatMessage

msgfmt_format_message

MessageFormatter::formatMessage -- msgfmt_format_message -- Quick format message

Description

Object oriented style

static string **MessageFormatter::formatMessage** (string `$locale`, string `$pattern`, array `$args`)

Procedural style

string **msgfmt_format_message** (string `$locale`, string `$pattern`, array `$args`)

Quick formatting function that formats the string without having to explicitly create the formatter object. Use this function when the format operation is done only once and does not need and parameters or state to be kept.

Parameters

`locale`
The locale to use for formatting locale-dependent parts

`pattern`
The pattern string to insert things into. The pattern uses an 'apostrophe-friendly' syntax; it is run through » umsg_autoQuoteApostrophe before being interpreted.

`args`
The array of values to insert into the format string

Return Values

The formatted pattern string or **FALSE** if an error occurred

Examples

Example #102 - msgfmt_format_message() example

```
<?php
echo msgfmt_format_message("en_US", "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree", array(4560, 123,
4560/123));
```

```
echo msgfmt_format_message("de", "{0,number,integer} Affen über
{1,number,integer} Bäume um {2,number} Affen pro Baum", array(4560, 123,
4560/123));
```

Example #103 - OO example

```
<?php
echo msgfmt_format_message("en_US", "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree", array(4560, 123,
4560/123));
echo msgfmt_format_message("de", "{0,number,integer} Affen über
{1,number,integer} Bäume um {2,number} Affen pro Baum", array(4560, 123,
4560/123));
?>
```

The above example will output:

```
4,560 monkeys on 123 trees make 37.073 monkeys per tree
4.560 Affen über 123 Bäume um 37,073 Affen pro Baum
```

See Also

- msgfmt_create()
- msgfmt_parse()
- msgfmt_get_error_code()
- msgfmt_get_error_message()

MessageFormatter::format

msgfmt_format

MessageFormatter::format -- msgfmt_format -- Format the message

Description

Object oriented style

string **MessageFormatter::format** (array `$args`)

Procedural style

string **msgfmt_format** (MessageFormatter `$fmt`, array `$args`)

Format the message by substituting the data into the format string according to the locale rules

Parameters

`fmt`
: The message formatter

`args`
: Arguments to insert into the format string

Return Values

The formatted string, or **FALSE** if an error occurred

Examples

Example #104 - msgfmt_format() example

```
<?php
$fmt = msgfmt_create("en_US", "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree");
echo msgfmt_format($fmt, array(4560, 123, 4560/123));
$fmt = msgfmt_create("de", "{0,number,integer} Affen über {1,number,integer}
Bäume um {2,number} Affen pro Baum");
echo msgfmt_format($fmt, array(4560, 123, 4560/123));
?>
```

Example #105 - OO example

```
<?php
$fmt = new MessageFormatter("en_US", "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree");
echo $fmt->format(array(4560, 123, 4560/123));
$fmt = new MessageFormatter("de", "{0,number,integer} Affen über
{1,number,integer} Bäume um {2,number} Affen pro Baum");
echo $fmt->format(array(4560, 123, 4560/123));
?>
```

The above example will output:

```
4,560 monkeys on 123 trees make 37.073 monkeys per tree
4.560 Affen über 123 Bäume um 37,073 Affen pro Baum
```

See Also

- msgfmt_create()
- msgfmt_parse()
- msgfmt_format_message()
- msgfmt_get_error_code()
- msgfmt_get_error_message()

MessageFormatter::getErrorCode

msgfmt_get_error_code

MessageFormatter::getErrorCode -- msgfmt_get_error_code -- Get the error code from last operation

Description

Object oriented style

integer **MessageFormatter::getErrorCode** (void)

Procedural style

integer **msgfmt_get_error_code** (MessageFormatter `$fmt`)

Get the error code from last operation.

Parameters

`fmt`
The message formatter

Return Values

The error code, one of UErrorCode values. Initial value is U_ZERO_ERROR.

Examples

Example #106 - msgfmt_get_error_code() example

```
<?php
$fmt = msgfmt_create("en_US", "{0, number} monkeys on {1, number} trees");
$str = msgfmt_format($fmt, array());
if(!$str) {
   echo "ERROR: ".msgfmt_get_error_message($fmt) . " (" .
msgfmt_get_error_code($fmt) . ")\n";
}
?>
```

Example #107 - OO example

```
<?php
```

```
$fmt = new MessageFormatter("en_US", "{0, number} monkeys on {1, number}
trees");
$str = $fmt->format(array());
if(!$str) {
   echo "ERROR: ".$fmt->getErrorMessage() . " (" . $fmt->getErrorCode() .
")\n";
}
?>
```

The above example will output:

```
ERROR: msgfmt_format: not enough parameters: U_ILLEGAL_ARGUMENT_ERROR (1)
```

See Also

- msgfmt_get_error_message()
- intl_get_error_code()
- intl_is_failure()

MessageFormatter::getErrorMessage

msgfmt_get_error_message

MessageFormatter::getErrorMessage -- msgfmt_get_error_message -- Get the error text from the last operation

Description

Object oriented style

string **MessageFormatter::getErrorMessage** (void)

Procedural style

string **msgfmt_get_error_message** (MessageFormatter `$fmt`)

Get the error text from the last operation.

Parameters

`fmt`
The message formatter

Return Values

Description of the last error.

Examples

Example #108 - msgfmt_get_error_message() example

```
<?php
$fmt = msgfmt_create("en_US", "{0, number} monkeys on {1, number} trees");
$str = msgfmt_format($fmt, array());
if(!$str) {
   echo "ERROR: ".msgfmt_get_error_message($fmt) . " (" .
msgfmt_get_error_code($fmt) . ")\n";
}
?>
```

Example #109 - OO example

```
<?php
```

```
$fmt = new MessageFormatter("en_US", "{0, number} monkeys on {1, number}
trees");
$str = $fmt->format(array());
if(!$str) {
   echo "ERROR: ".$fmt->getErrorMessage() . " (" . $fmt->getErrorCode() .
")\n";
}
?>
```

The above example will output:

```
ERROR: msgfmt_format: not enough parameters: U_ILLEGAL_ARGUMENT_ERROR (1)
```

See Also

- msgfmt_get_error_code()
- intl_get_error_code()
- intl_is_failure()

MessageFormatter::getLocale

msgfmt_get_locale

MessageFormatter::getLocale -- msgfmt_get_locale -- Get the locale for which the formatter was created.

Description

Object oriented style

string **MessageFormatter::getLocale** (void)

Procedural style

string **msgfmt_get_locale** (NumberFormatter `$formatter`)

Get the locale for which the formatter was created.

Parameters

`formatter`
The formatter resource

Return Values

The locale name

Examples

Example #110 - msgfmt_get_locale() example

```
<?php
$fmt = msgfmt_create('en_US', "Number {0,number}");
echo msgfmt_get_locale($fmt);
?>
```

Example #111 - OO example

```
<?php
$fmt = new MessageFormatter('en_US', "Number {0,number}");
echo $fmt->getLocale();
?>
```

The above example will output:

```
en_US
```

See Also

- msgfmt_create()

MessageFormatter::getPattern

msgfmt_get_pattern

MessageFormatter::getPattern -- msgfmt_get_pattern -- Get the pattern used by the formatter

Description

Object oriented style

string **MessageFormatter::getPattern** (void)

Procedural style

string **msgfmt_get_pattern** (MessageFormatter `$fmt`)

Get the pattern used by the formatter

Parameters

`fmt`
The message formatter

Return Values

The pattern string for this message formatter

Examples

Example #112 - msgfmt_get_pattern() example

```
<?php
$fmt = msgfmt_create( "en_US", "{0, number} monkeys on {1, number} trees" );
echo "Default pattern: '" . msgfmt_get_pattern( $fmt ) . "'\n";
echo "Formatting result: " . msgfmt_format( $fmt, array(123, 456) ) . "\n";

msgfmt_set_pattern( $fmt, "{0, number} trees hosting {1, number} monkeys" );
echo "New pattern: '" . msgfmt_get_pattern( $fmt ) . "'\n";
echo "Formatted number: " . msgfmt_format( $fmt, array(123, 456) ) . "\n";
?>
```

Example #113 - OO example

```
<?php
$fmt = new MessageFormatter( "en_US", "{0, number} monkeys on {1, number}
trees" );
echo "Default pattern: '" . $fmt->getPattern() . "'\n";
echo "Formatting result: " . $fmt->format(array(123, 456)) . "\n";

$fmt->setPattern("{0, number} trees hosting {1, number} monkeys" );
echo "New pattern: '" . $fmt->getPattern() . "'\n";
echo "Formatted number: " . $fmt->format(array(123, 456)) . "\n";
?>
```

The above example will output:

```
Default pattern: '{0,number} monkeys on {1,number} trees'
Formatting result: 123 monkeys on 456 trees
New pattern: '{0,number} trees hosting {1,number} monkeys'
Formatted number: 123 trees hosting 456 monkeys
```

See Also

- msgfmt_create()
- msgfmt_set_pattern()

MessageFormatter::parseMessage

msgfmt_parse_message

MessageFormatter::parseMessage -- msgfmt_parse_message -- Quick parse input string

Description

Object oriented style

static array **MessageFormatter::parseMessage** (string `$locale`, string `$value`)

Procedural style

array **msgfmt_parse_message** (string `$locale`, string `$value`)

Parses input string without explicitly creating the formatter object. Use this function when the format operation is done only once and does not need and parameters or state to be kept.

Parameters

`locale`
The locale to use for parsing locale-dependent parts

`value`
The string to parse for items

Return Values

An array containing items extracted, or **FALSE** on error

Examples

Example #114 - msgfmt_parse_message() example

```
<?php
$fmt = msgfmt_parse_message('en_US', "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree",
                            "4,560 monkeys on 123 trees make 37.073 monkeys
per tree");
var_export($res);

$fmt = msgfmt_parse_message('de', "{0,number,integer} Affen über
{1,number,integer} Bäume um {2,number} Affen pro Baum",
                            "4.560 Affen über 123 Bäume um 37,073 Affen pro
```

```
Baum");
var_export($res);
?>
```

Example #115 - OO example

```
<?php
$fmt = MessageFormatter::parseMessage('en_US', "{0,number,integer} monkeys
on {1,number,integer} trees make {2,number} monkeys per tree",
                            "4,560 monkeys on 123 trees make 37.073 monkeys
per tree");
var_export($res);

$fmt = MessageFormatter::parseMessage('de', "{0,number,integer} Affen über
{1,number,integer} Bäume um {2,number} Affen pro Baum",
                            "4.560 Affen über 123 Bäume um 37,073 Affen pro
Baum");
var_export($res);
?>
```

The above example will output:

```
array (
 0 => 4560,
 1 => 123,
 2 => 37.073,
)
array (
 0 => 4560,
 1 => 123,
 2 => 37.073,
)
```

See Also

- msgfmt_create()
- msgfmt_format_message()
- msgfmt_parse()

MessageFormatter::parse

msgfmt_parse

MessageFormatter::parse -- msgfmt_parse -- Parse input string according to pattern

Description

Object oriented style

array **MessageFormatter::parse** (string `$value`)

Procedural style

array **msgfmt_parse** (MessageFormatter `$fmt`, string `$value`)

Parses input string and return any extracted items as an array.

Parameters

`fmt`
The message formatter

`value`
The string to parse

Return Values

An array containing the items extracted, or **FALSE** on error

Examples

Example #116 - msgfmt_parse() example

```
<?php
$fmt = msgfmt_create('en_US', "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree");
$res = msgfmt_parse($fmt, "4,560 monkeys on 123 trees make 37.073 monkeys
per tree");
var_export($res);

$fmt = msgfmt_create('de', "{0,number,integer} Affen über {1,number,integer}
Bäume um {2,number} Affen pro Baum");
$res = msgfmt_parse($fmt, "4.560 Affen über 123 Bäume um 37,073 Affen pro
Baum");
var_export($res);
```

```
?>
```

Example #117 - OO example

```
<?php
$fmt = new MessageFormatter('en_US', "{0,number,integer} monkeys on
{1,number,integer} trees make {2,number} monkeys per tree");
$res = $fmt->parse("4,560 monkeys on 123 trees make 37.073 monkeys per
tree");
var_export($res);

$fmt = new MessageFormatter('de', "{0,number,integer} Affen über
{1,number,integer} Bäume um {2,number} Affen pro Baum");
$res = $fmt->parse("4.560 Affen über 123 Bäume um 37,073 Affen pro Baum");
var_export($res);
?>
```

The above example will output:

```
array (
 0 => 4560,
 1 => 123,
 2 => 37.073,
)
array (
 0 => 4560,
 1 => 123,
 2 => 37.073,
)
```

See Also

- msgfmt_create()
- msgfmt_format()
- msgfmt_parse_message()

MessageFormatter::setPattern

msgfmt_set_pattern

MessageFormatter::setPattern -- msgfmt_set_pattern -- Set the pattern used by the formatter

Description

Object oriented style

boolean **MessageFormatter::setPattern** (string `$pattern`)

Procedural style

boolean **msgfmt_set_pattern** (MessageFormatter `$fmt`, string `$pattern`)

Set the pattern used by the formatter

Parameters

`fmt`
The message formatter

`pattern`
The pattern string to use in this message formatter. The pattern uses an 'apostrophe-friendly' syntax; it is run through » umsg_autoQuoteApostrophe before being interpreted.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #118 - msgfmt_set_pattern() example

```
<?php
$fmt = msgfmt_create( "en_US", "{0, number} monkeys on {1, number} trees" );
echo "Default pattern: '" . msgfmt_get_pattern( $fmt ) . "'\n";
echo "Formatting result: " . msgfmt_format( $fmt, array(123, 456) ) . "\n";

msgfmt_set_pattern( $fmt, "{0, number} trees hosting {1, number} monkeys" );
echo "New pattern: '" . msgfmt_get_pattern( $fmt ) . "'\n";
echo "Formatted number: " . msgfmt_format( $fmt, array(123, 456) ) . "\n";
?>
```

Example #119 - OO example

```
<?php
$fmt = new MessageFormatter( "en_US", "{0, number} monkeys on {1, number}
trees" );
echo "Default pattern: '" . $fmt->getPattern() . "'\n";
echo "Formatting result: " . $fmt->format(array(123, 456)) . "\n";

$fmt->setPattern("{0, number} trees hosting {1, number} monkeys" );
echo "New pattern: '" . $fmt->getPattern() . "'\n";
echo "Formatted number: " . $fmt->format(array(123, 456)) . "\n";
?>
```

The above example will output:

```
Default pattern: '{0,number} monkeys on {1,number} trees'
Formatting result: 123 monkeys on 456 trees
New pattern: '{0,number} trees hosting {1,number} monkeys'
Formatted number: 123 trees hosting 456 monkeys
```

See Also

- msgfmt_create()
- msgfmt_get_pattern()

The IntlDateFormatter class

Introduction

Date Formatter is a concrete class that enables locale-dependent formatting/parsing of dates using pattern strings and/or canned patterns.

This class represents the ICU date formatting functionality. It allows users to display dates in a localized format or to parse strings into PHP date values using pattern strings and/or canned patterns.

Class synopsis

IntlDateFormatter

IntlDateFormatter {

/* Methods */

IntlDateFormatter::__construct (string `$locale`, integer `$datetype`, integer `$timetype` [, string `$timezone` [, integer `$calendar` [, string `$pattern`]]])

static IntlDateFormatter **IntlDateFormatter::create** (string `$locale`, integer `$datetype`, integer `$timetype` [, string `$timezone` [, integer `$calendar` [, string `$pattern`]]])

string **IntlDateFormatter::format** (mixed `$value`)

integer **IntlDateFormatter::getCalendar** (void)

integer **IntlDateFormatter::getDateType** (void)

integer **IntlDateFormatter::getErrorCode** (void)

string **IntlDateFormatter::getErrorMessage** (void)

string **IntlDateFormatter::getLocale** ([integer `$which`])

string **IntlDateFormatter::getPattern** (void)

integer **IntlDateFormatter::getTimeType** (void)

string **IntlDateFormatter::getTimeZoneId** (void)

boolean **IntlDateFormatter::isLenient** (void)

array **IntlDateFormatter::localtime** (string `$value`, integer `$parse_pos`)

integer **IntlDateFormatter::parse** (string `$value`, integer `$parse_pos`)

boolean **IntlDateFormatter::setCalendar** (integer `$which`)

boolean **IntlDateFormatter::setLenient** (boolean `$lenient`)

boolean **IntlDateFormatter::setPattern** (string `$pattern`)

boolean **IntlDateFormatter::setTimeZoneId** (string `$zone`)
}

See Also

- » ICU Date formatter

Predefined Constants

These constants are used to specify different formats in the constructor for DateType and TimeType.

IntlDateFormatter::NONE (string)
Do not include this element

IntlDateFormatter::FULL (string)
Completely specified style (Tuesday, April 12, 1952 AD or 3:30:42pm PST)

IntlDateFormatter::LONG (string)
Long style (January 12, 1952 or 3:30:32pm)

IntlDateFormatter::MEDIUM (string)
Medium style (Jan 12, 1952)

IntlDateFormatter::SHORT (string)
Most abbreviated style, only essential data (12/13/52 or 3:30pm)

The following int constants are used to specify the calendar. These calendars are all based directly on the Gregorian calendar. Non-Gregorian calendars need to be specified in locale. Examples might include locale="hi@calendar=BUDDHIST".

IntlDateFormatter::TRADITIONAL (string)
Non-Gregorian Calendar

IntlDateFormatter::GREGORIAN (string)
Gregorian Calendar

IntlDateFormatter::create

datefmt_create

IntlDateFormatter::__construct

IntlDateFormatter::create -- datefmt_create -- IntlDateFormatter::__construct -- Create a date formatter

Description

Object oriented style

static IntlDateFormatter **IntlDateFormatter::create** (string `$locale`, integer `$datetype`, integer `$timetype` [, string `$timezone` [, integer `$calendar` [, string `$pattern`]]])

Object oriented style (constructor)

IntlDateFormatter::__construct (string `$locale`, integer `$datetype`, integer `$timetype` [, string `$timezone` [, integer `$calendar` [, string `$pattern`]]])

Procedural style

IntlDateFormatter **datefmt_create** (string `$locale`, integer `$datetype`, integer `$timetype` [, string `$timezone` [, integer `$calendar` [, string `$pattern`]]])

Create a date formatter

Parameters

`locale`
Locale to use when formatting or parsing.

`datetype`
Date type to use (none, short, medium, long, full). This is one of the IntlDateFormatter constants.

`timetype`
Time type to use (none, short, medium, long, full). This is one of the IntlDateFormatter constants.

`timezone`
Time zone ID, default is system default.

`calendar`
Calendar to use for formatting or parsing; default is Gregorian. This is one of the

IntlDateFormatter calendar constants.

pattern
Optional pattern to use when formatting or parsing

Return Values

Examples

Example #120 - datefmt_create() example

```
<?php
$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "First Formatted output is ".datefmt_format( $fmt , 0);
$fmt = datefmt_create( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "Second Formatted output is ".datefmt_format( $fmt , 0);

$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "First Formatted output with pattern is ".datefmt_format( $fmt , 0);
$fmt = datefmt_create( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "Second Formatted output with pattern is ".datefmt_format( $fmt , 0);
?>
```

Example #121 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "First Formatted output is ".$fmt->format(0);
$fmt = new IntlDateFormatter( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "Second Formatted output is ".$fmt->format(0);

$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "First Formatted output with pattern is ".$fmt->format(0);
$fmt = new IntlDateFormatter( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN , "MM/dd/yyyy");
echo "Second Formatted output with pattern is ".$fmt->format(0);
```

```
?>
```

The above example will output:

```
First Formatted output is Wednesday, December 31, 1969 4:00:00 PM PT
Second Formatted output is Mittwoch, 31. Dezember 1969 16:00 Uhr GMT-08:00
First Formatted output with pattern is 12/31/1969
Second Formatted output with pattern is 12/31/1969
```

See Also

- datefmt_format()
- datefmt_parse()
- datefmt_get_error_code()
- datefmt_get_error_message()

IntlDateFormatter::format

datefmt_format

IntlDateFormatter::format -- datefmt_format -- Format the date/time value as a string

Description

Object oriented style

string **IntlDateFormatter::format** (mixed `$value`)

Procedural style

string **datefmt_format** (IntlDateFormatter `$fmt`, mixed `$value`)

Formats the time value as a string.

Parameters

`fmt`
: The date formatter resource.

`value`
: Value to format. Can be integer for an Unix timestamp value (seconds since epoch, UTC) or array for a localtime() array.

Return Values

The formatted string or, if an error occurred, **FALSE**.

Examples

Example #122 - datefmt_format() example

```
<?php
$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "First Formatted output is ".datefmt_format( $fmt , 0);
$fmt = datefmt_create( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "Second Formatted output is ".datefmt_format( $fmt , 0);

$fmt = datefmt_create( "en_US"
```

```
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "First Formatted output with pattern is ".datefmt_format( $fmt , 0);
$fmt = datefmt_create( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "Second Formatted output with pattern is ".datefmt_format( $fmt , 0);
?>
```

Example #123 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "First Formatted output is ".$fmt->format(0);
$fmt = new IntlDateFormatter( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "Second Formatted output is ".$fmt->format(0);

$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "First Formatted output with pattern is ".$fmt->format(0);
$fmt = new IntlDateFormatter( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN , "MM/dd/yyyy");
echo "Second Formatted output with pattern is ".$fmt->format(0);
?>
```

The above example will output:

```
First Formatted output is Wednesday, December 31, 1969 4:00:00 PM PT
Second Formatted output is Mittwoch, 31. Dezember 1969 16:00 Uhr GMT-08:00
First Formatted output with pattern is 12/31/1969
Second Formatted output with pattern is 12/31/1969
```

See Also

- datefmt_create()
- datefmt_parse()
- datefmt_get_error_code()
- datefmt_get_error_message()

IntlDateFormatter::getCalendar

datefmt_get_calendar

IntlDateFormatter::getCalendar -- datefmt_get_calendar -- Get the calendar used for the IntlDateFormatter

Description

Object oriented style

integer **IntlDateFormatter::getCalendar** (void)

Procedural style

integer **datefmt_get_calendar** (IntlDateFormatter `$fmt`)

Parameters

`fmt`
The formatter resource

Return Values

The calendar being used by the formatter.

Examples

Example #124 - datefmt_get_calendar() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "calendar of the formatter is : ".datefmt_get_calendar($fmt);
datefmt_set_calendar($fmt,IntlDateFormatter::TRADITIONAL);
echo "Now calendar of the formatter is : ".datefmt_get_calendar($fmt);
?>
```

Example #125 - OO example

```
<?php
```

```
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "calendar of the formatter is : ".$fmt->getCalendar();
$fmt->setCalendar(IntlDateFormatter::TRADITIONAL);
echo "Now calendar of the formatter is : ".$fmt->getCalendar();

?>
```

The above example will output:

```
calendar of the formatter is : 1
Now calendar of the formatter is : 0
```

See Also

- datefmt_set_calendar()
- datefmt_create()

IntlDateFormatter::getDateType

datefmt_get_datetype

IntlDateFormatter::getDateType -- datefmt_get_datetype -- Get the datetype used for the IntlDateFormatter

Description

Object oriented style

integer **IntlDateFormatter::getDateType** (void)

Procedural style

integer **datefmt_get_datetype** (IntlDateFormatter `$fmt`)

Returns date type used by the formatter.

Parameters

`fmt`
The formatter resource.

Return Values

The current date type value of the formatter.

Examples

Example #126 - datefmt_get_datetype() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "datetype of the formatter is : ".datefmt_get_datetype($fmt);
echo "First Formatted output with datetype is ".datefmt_format( $fmt , 0);
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::SHORT,IntlDateFormatter::FULL,'America/Los_Angeles',Intl
DateFormatter::GREGORIAN  );
echo "Now datetype of the formatter is : ".datefmt_get_datetype($fmt);
echo "Second Formatted output with datetype is ".datefmt_format( $fmt , 0);

?>
```

Example #127 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN);
echo "datetype of the formatter is : ".$fmt->getDateType();
echo "First Formatted output is ".datefmt_format( $fmt , 0);
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::SHORT,IntlDateFormatter::FULL,'America/Los_Angeles',Intl
DateFormatter::GREGORIAN);
echo "Now datetype of the formatter is : ".$fmt->getDateType();
echo "Second Formatted output is ".datefmt_format( $fmt , 0);

?>
```

The above example will output:

```
datetype of the formatter is : 0
First Formatted output is Wednesday, December 31, 1969 4:00:00 PM PT
Now datetype of the formatter is : 2
Second Formatted output is 12/31/69 4:00:00 PM PT
```

See Also

- datefmt_get_timetype()
- datefmt_create()

IntlDateFormatter::getErrorCode

datefmt_get_error_code

IntlDateFormatter::getErrorCode -- datefmt_get_error_code -- Get the error code from last operation

Description

Object oriented style

integer **IntlDateFormatter::getErrorCode** (void)

Procedural style

integer **datefmt_get_error_code** (IntlDateFormatter `$fmt`)

Get the error code from last operation. Returns error code from the last number formatting operation.

Parameters

`fmt`
The formatter resource.

Return Values

The error code, one of UErrorCode values. Initial value is U_ZERO_ERROR.

Examples

Example #128 - datefmt_get_error_code() example

```
<?php
$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
$str = datefmt_format($fmt);
if(!$str) {
   echo "ERROR: ".datefmt_get_error_message($fmt) . " (" .
datefmt_get_error_code($fmt) . ")\n";
}
?>
```

Example #129 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
$str = $fmt->format();
if(!$str) {
   echo "ERROR: ".$fmt->getErrorMessage() . " (" . $fmt->getErrorCode() .
")\n";
}
?>
```

The above example will output:

```
ERROR: U_ZERO_ERROR (0)
```

See Also

- datefmt_get_error_message()
- intl_get_error_code()
- intl_is_failure()

IntlDateFormatter::getErrorMessage

datefmt_get_error_message

IntlDateFormatter::getErrorMessage -- datefmt_get_error_message -- Get the error text from the last operation.

Description

Object oriented style

string **IntlDateFormatter::getErrorMessage** (void)

Procedural style

string **datefmt_get_error_message** (IntlDateFormatter `$fmt`)

Get the error text from the last operation.

Parameters

`fmt`
The formatter resource.

Return Values

Description of the last error.

Examples

Example #130 - datefmt_get_error_message() **example**

```
<?php
$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
$str = datefmt_format($fmt);
if(!$str) {
   echo "ERROR: ".datefmt_get_error_message($fmt) . " (" .
datefmt_get_error_code($fmt) . ")\n";
}
?>
```

Example #131 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
$str = $fmt->format();
if(!$str) {
   echo "ERROR: ".$fmt->getErrorMessage() . " (" . $fmt->getErrorCode() .
")\n";
}

?>
```

The above example will output:

```
ERROR: U_ZERO_ERROR (0)
```

See Also

- datefmt_get_error_code()
- intl_get_error_code()
- intl_is_failure()

IntlDateFormatter::getLocale

datefmt_get_locale

IntlDateFormatter::getLocale -- datefmt_get_locale -- Get the locale used by formatter

Description

Object oriented style

string **IntlDateFormatter::getLocale** ([integer `$which`])

Procedural style

string **datefmt_get_locale** (IntlDateFormatter `$fmt` [, integer `$which`])

Get locale used by the formatter.

Parameters

`fmt`
: The formatter resource

`hich`
: You can choose between valid and actual locale (**Locale::VALID_LOCALE**, **Locale::ACTUAL_LOCALE**, respectively). The default is the actual locale.

Return Values

the locale of this formatter or 'false' if error

Examples

Example #132 - datefmt_get_locale() example

```
<?php
$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "locale of the formatter is : ".datefmt_get_locale($fmt);
echo "First Formatted output is ".datefmt_format( $fmt , 0);
echo "locale of the formatter is : ".datefmt_get_locale($fmt);
$fmt = datefmt_create( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "Second Formatted output is ".datefmt_format( $fmt , 0);
```

```
?>
```

Example #133 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "locale of the formatter is : ".$fmt->getLocale();
echo "First Formatted output is ".$fmt->format(0);
$fmt = new IntlDateFormatter( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "locale of the formatter is : ".$fmt->getLocale();
echo "Second Formatted output is ".$fmt->format(0);

?>
```

The above example will output:

```
locale of the formatter is : en
First Formatted output is Wednesday, December 31, 1969 4:00:00 PM PT
locale of the formatter is : de
Second Formatted output is Mittwoch, 31. Dezember 1969 16:00 Uhr GMT-08:00
```

See Also

- datefmt_create()

IntlDateFormatter::getPattern

datefmt_get_pattern

IntlDateFormatter::getPattern -- datefmt_get_pattern -- Get the pattern used for the IntlDateFormatter

Description

Object oriented style

string **IntlDateFormatter::getPattern** (void)

Procedural style

string **datefmt_get_pattern** (IntlDateFormatter `$fmt`)

Get pattern used by the formatter.

Parameters

`fmt`
The formatter resource.

Return Values

The pattern string being used to format/parse.

Examples

Example #134 - datefmt_get_pattern() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "pattern of the formatter is : ".datefmt_get_pattern($fmt);
echo "First Formatted output with pattern is ".datefmt_format( $fmt , 0);
datefmt_set_pattern($fmt,'yyyymmdd hh:mm:ss z');
echo "Now pattern of the formatter is : ".datefmt_get_pattern($fmt);
echo "Second Formatted output with pattern is ".datefmt_format( $fmt , 0);

?>
```

Example #135 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN,"MM/dd/yyyy"  );
echo "pattern of the formatter is : ".$fmt->getPattern();
echo "First Formatted output is ".datefmt_format( $fmt , 0);
$fmt->setPattern('yyyymmdd hh:mm:ss z');
echo "Now pattern of the formatter is : ".$fmt->getPattern();
echo "Second Formatted output is ".datefmt_format( $fmt , 0);
?>
```

The above example will output:

```
pattern of the formatter is : MM/dd/yyyy
First Formatted output is 12/31/1969
Now pattern of the formatter is : yyyymmdd hh:mm:ss z
Second Formatted output is 19690031 04:00:00 PST
```

See Also

- datefmt_set_pattern()
- datefmt_create()

IntlDateFormatter::getTimeType

datefmt_get_timetype

IntlDateFormatter::getTimeType -- datefmt_get_timetype -- Get the timetype used for the IntlDateFormatter

Description

Object oriented style

integer **IntlDateFormatter::getTimeType** (void)

Procedural style

integer **datefmt_get_timetype** (IntlDateFormatter `$fmt`)

Return time type used by the formatter.

Parameters

`fmt`
The formatter resource.

Return Values

The current date type value of the formatter.

Examples

Example #136 - datefmt_get_timetype() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "timetype of the formatter is : ".datefmt_get_timetype($fmt);
echo "First Formatted output with timetype is ".datefmt_format( $fmt , 0);
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::SHORT,'America/Los_Angeles',Intl
DateFormatter::GREGORIAN  );
echo "Now timetype of the formatter is : ".datefmt_get_timetype($fmt);
echo "Second Formatted output with timetype is ".datefmt_format( $fmt , 0);

?>
```

Example #137 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN);
echo "timetype of the formatter is : ".$fmt->getTimeType();
echo "First Formatted output is ".datefmt_format( $fmt , 0);
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::SHORT,'America/Los_Angeles',Intl
DateFormatter::GREGORIAN);
echo "Now timetype of the formatter is : ".$fmt->getTimeType();
echo "Second Formatted output is ".datefmt_format( $fmt , 0);

?>
```

The above example will output:

```
timetype of the formatter is : 0
First Formatted output is Wednesday, December 31, 1969 4:00:00 PM PT
Now timetype of the formatter is : 3
Second Formatted output is Wednesday, December 31, 1969 4:00 PM
```

See Also

- datefmt_get_datetype()
- datefmt_create()

IntlDateFormatter::getTimeZoneId

datefmt_get_timezone_id

IntlDateFormatter::getTimeZoneId -- datefmt_get_timezone_id -- Get the timezone-id used for the IntlDateFormatter

Description

Object oriented style

string **IntlDateFormatter::getTimeZoneId** (void)

Procedural style

string **datefmt_get_timezone_id** (IntlDateFormatter `$fmt`)

Get the timezone-id used for the IntlDateFormatter.

Parameters

`fmt`
The formatter resource.

Return Values

ID string for the time zone used by this formatter.

Examples

Example #138 - datefmt_get_timezone_id() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "timezone_id of the formatter is : ".datefmt_get_timezone_id($fmt);
datefmt_set_timezone_id($fmt,'CN');
echo "Now timezone_id of the formatter is : ".datefmt_get_timezone_id($fmt);

?>
```

Example #139 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "timezone_id of the formatter is : ".$fmt->getTimezoneId();
$fmt->setTimezoneId('CN');
echo "Now timezone_id of the formatter is : ".$fmt->getTimezoneId();

?>
```

The above example will output:

```
timezone_id of the formatter is : America/Los_Angeles
Now timezone_id of the formatter is : CN
```

See Also

- datefmt_set_timezone_id()
- datefmt_create()

IntlDateFormatter::isLenient

datefmt_is_lenient

IntlDateFormatter::isLenient -- datefmt_is_lenient -- Get the lenient used for the IntlDateFormatter

Description

Object oriented style

boolean **IntlDateFormatter::isLenient** (void)

Procedural style

boolean **datefmt_is_lenient** (IntlDateFormatter `$fmt`)

Check if the parser is strict or lenient in interpretting inputs that do not match the pattern exactly.

Parameters

`fmt`
The formatter resource.

Return Values

TRUE if parser is lenient, **FALSE** if parser is strict. By default the parser is strict.

Examples

Example #140 - datefmt_is_lenient() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGO
RIAN  ,"dd/mm/yyyy");
echo "lenient of the formatter is : ";
if( $fmt->isLenient() ){
       echo('TRUE');
}else{
       echo('FALSE');
}
datefmt_parse($fmt,"35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
```

```
.datefmt_parse($fmt,"35/13/1971");
if( intl_get_error_code() !=0 ){
       echo "Error_msg is : ".intl_get_error_message();
       echo "Error_code is : ".intl_get_error_code();
}
datefmt_set_lenient($fmt,false);
echo "Now lenient of the formatter is : ";
if( $fmt->isLenient() ){
       echo('TRUE');
}else{
       echo('FALSE');
}
datefmt_parse($fmt,"35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
.datefmt_parse($fmt,"35/13/1971");
if( intl_get_error_code() !=0 ){
       echo "Error_msg is : ".intl_get_error_message();
       echo "Error_code is : ".intl_get_error_code();
}

?>
```

Example #141 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GRE
GORIAN,"dd/mm/yyyy"  );
echo "lenient of the formatter is : ";
if( $fmt->isLenient() ){
       echo('TRUE');
}else{
       echo('FALSE');
}
$fmt->parse("35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
.$fmt->parse("35/13/1971");
if( intl_get_error_code() !=0 ){
       echo "Error_msg is : ".intl_get_error_message();
       echo "Error_code is : ".intl_get_error_code();
}

$fmt->setLenient(FALSE);
echo "Now lenient of the formatter is : ";
if( $fmt->isLenient() ){
       echo('TRUE');
}else{
       echo('FALSE');
}
$fmt->parse("35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
.$fmt->parse("35/13/1971");
if( intl_get_error_code() !=0 ){
       echo "Error_msg is : ".intl_get_error_message();
       echo "Error_code is : ".intl_get_error_code();
}
```

```
?>
```

The above example will output:

```
lenient of the formatter is : TRUE
Trying to do parse('35/13/1971').
Result is : -2147483
Now lenient of the formatter is : FALSE
Trying to do parse('35/13/1971').
Result is : Error_msg is : Date parsing failed: U_PARSE_ERROR Error_code is : 9
```

See Also

- datefmt_set_lenient()
- datefmt_create()

IntlDateFormatter::localtime

datefmt_localtime

IntlDateFormatter::localtime -- datefmt_localtime -- Parse string to a field-based time value

Description

Object oriented style

array **IntlDateFormatter::localtime** (string `$value`, integer `$parse_pos`)

Procedural style

array **datefmt_localtime** (IntlDateFormatter `$fmt`, string `$value`, integer `$parse_pos`)

Converts string $value to a field-based time value (an array of various fields), starting at $parse_pos and consuming as much of the input value as possible.

Parameters

`fmt`
: The formatter resource

`value`
: string to convert to a time

`parse_pos`
: Position at which to start the parsing in $value (zero-based). If no error occurs before $value is consumed, $parse_pos will contain -1 otherwise it will contain the position at which parsing ended . If $parse_pos > strlen($value), the parse fails immediately.

Return Values

Localtime compatible array of integers : contains 24 hour clock value in tm_hour field

Examples

Example #142 - datefmt_localtime() **example**

```
<?php

$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
```

```
$arr = datefmt_localtime( $fmt, "Wednesday, December 31, 1969 4:00:00 PM
PT",0);
echo "First parsed output is ";
if ($arr) {
   foreach ($arr as $key => $value) {
       echo "$key : $value , ";
   }
}

?>
```

Example #143 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
$arr = $fmt->localtime( "Wednesday, December 31, 1969 4:00:00 PM PT",0);
echo "First parsed output is ";
if ($arr) {
   foreach ($arr as $key => $value) {
       echo "$key : $value , ";
   }
}

?>
```

The above example will output:

```
First parsed output is tm_sec : 0 , tm_min : 0 , tm_hour : 16 , tm_year : 1969 ,

tm_mday : 31 , tm_wday : 4 , tm_yday : 365 , tm_mon : 11 , tm_isdst : 0 ,
```

See Also

- datefmt_create()
- datefmt_format()
- datefmt_parse()
- datefmt_get_error_code()
- datefmt_get_error_message()

IntlDateFormatter::parse

datefmt_parse

IntlDateFormatter::parse -- datefmt_parse -- Parse string to a timestamp value

Description

Object oriented style

integer **IntlDateFormatter::parse** (string `$value`, integer `$parse_pos`)

Procedural style

integer **datefmt_parse** (IntlDateFormatter `$fmt`, string `$value`, integer `$parse_pos`)

Converts string $value to an incremental time value, starting at $parse_pos and consuming as much of the input value as possible.

Parameters

`fmt`
: The formatter resource

`value`
: string to convert to a time

`parse_pos`
: Position at which to start the parsing in $value (zero-based). If no error occurs before $value is consumed, $parse_pos will contain -1 otherwise it will contain the position at which parsing ended (and the error occurred). This variable will contain the end position if the parse fails. If $parse_pos > strlen($value), the parse fails immediately.

Return Values

timestamp parsed value

Examples

Example #144 - datefmt_parse() example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
```

```
echo "First parsed output is ".$fmt->parse("Wednesday, December 31, 1969
4:00:00 PM PT");
$fmt = new IntlDateFormatter( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "Second parsed output is ".$fmt->parse("Mittwoch, 31. Dezember 1969
16:00 Uhr GMT-08:00");
?>
```

Example #145 - OO example

```
<?php
$fmt = datefmt_create( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "First parsed output is ".datefmt_parse( $fmt ,  "Wednesday, December
20, 1989 4:00:00 PM PT");
$fmt = datefmt_create( "de-DE"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "Second parsed output is ".datefmt_parse( $fmt , "Mittwoch, 20.
Dezember 1989 16:00 Uhr GMT-08:00");
?>
```

The above example will output:

```
First parsed output is 630201600
Second parsed output is 630201600
```

See Also

- datefmt_create()
- datefmt_format()
- datefmt_localtime()
- datefmt_get_error_code()
- datefmt_get_error_message()

IntlDateFormatter::setCalendar

datefmt_set_calendar

IntlDateFormatter::setCalendar -- datefmt_set_calendar -- sets the calendar used to the appropriate calendar, which must be

Description

Object oriented style

boolean **IntlDateFormatter::setCalendar** (integer `$which`)

Procedural style

boolean **datefmt_set_calendar** (IntlDateFormatter `$fmt`, integer `$which`)

Sets the calendar used by the formatter.

Parameters

`fmt`
The formatter resource.

`which`
The calendar to use. Default is **IntlDateFormatter::GREGORIAN**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #146 - datefmt_set_calendar() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "calendar of the formatter is : ".datefmt_get_calendar($fmt);
datefmt_set_calendar($fmt,IntlDateFormatter::TRADITIONAL);
echo "Now calendar of the formatter is : ".datefmt_get_calendar($fmt);
?>
```

Example #147 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "calendar of the formatter is : ".$fmt->getCalendar();
$fmt->setCalendar(IntlDateFormatter::TRADITIONAL);
echo "Now calendar of the formatter is : ".$fmt->getCalendar();
?>
```

The above example will output:

```
calendar of the formatter is : 1
Now calendar of the formatter is : 0
```

See Also

- datefmt_get_calendar()
- datefmt_create()

IntlDateFormatter::setLenient

datefmt_set_lenient

IntlDateFormatter::setLenient -- datefmt_set_lenient -- Set the leniency of the parser

Description

Object oriented style

boolean **IntlDateFormatter::setLenient** (boolean `$lenient`)

Procedural style

boolean **datefmt_set_lenient** (IntlDateFormatter `$fmt`, boolean `$lenient`)

Define if the parser is strict or lenient in interpretting inputs that do not match the pattern exactly. Enabling lenient parsing allows the parser to accept otherwise flawed date or time patterns, parsing as much as possible to obtain a value. Extra space, unrecognized tokens, or invalid values ("Feburary 30th") are not accepted.

Parameters

`fmt`
: The formatter resource

`lenient`
: Sets whether the parser is lenient or not, default is **FALSE** (strict).

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #148 - datefmt_set_lenient() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGO
RIAN  ,"dd/mm/yyyy");
echo "lenient of the formatter is : ";
if( $fmt->isLenient() ){
       echo('TRUE');
}else{
```

```
        echo('FALSE');
}
datefmt_parse($fmt,"35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
.datefmt_parse($fmt,"35/13/1971");
if( intl_get_error_code() !=0 ){
        echo "Error_msg is : ".intl_get_error_message();
        echo "Error_code is : ".intl_get_error_code();
}
datefmt_set_lenient($fmt,false);
echo "Now lenient of the formatter is : ";
if( $fmt->isLenient() ){
        echo('TRUE');
}else{
        echo('FALSE');
}
datefmt_parse($fmt,"35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
.datefmt_parse($fmt,"35/13/1971");
if( intl_get_error_code() !=0 ){
        echo "Error_msg is : ".intl_get_error_message();
        echo "Error_code is : ".intl_get_error_code();
}

?>
```

Example #149 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GRE
GORIAN,"dd/mm/yyyy"  );
echo "lenient of the formatter is : ";
if( $fmt->isLenient() ){
        echo('TRUE');
}else{
        echo('FALSE');
}
$fmt->parse("35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
.$fmt->parse("35/13/1971");
if( intl_get_error_code() !=0 ){
        echo "Error_msg is : ".intl_get_error_message();
        echo "Error_code is : ".intl_get_error_code();
}

$fmt->setLenient(FALSE);
echo "Now lenient of the formatter is : ";
if( $fmt->isLenient() ){
        echo('TRUE');
}else{
        echo('FALSE');
}
$fmt->parse("35/13/1971");
echo "\n Trying to do parse('35/13/1971').Result is : "
.$fmt->parse("35/13/1971");
if( intl_get_error_code() !=0 ){
```

```
        echo "Error_msg is : ".intl_get_error_message();
        echo "Error_code is : ".intl_get_error_code();
}

?>
```

The above example will output:

```
lenient of the formatter is : TRUE
Trying to do parse('35/13/1971').
Result is : -2147483
Now lenient of the formatter is : FALSE
Trying to do parse('35/13/1971').
Result is : Error_msg is : Date parsing failed: U_PARSE_ERROR Error_code is : 9
```

See Also

- datefmt_is_lenient()
- datefmt_create()

IntlDateFormatter::setPattern

datefmt_set_pattern

IntlDateFormatter::setPattern -- datefmt_set_pattern -- Set the pattern used for the IntlDateFormatter

Description

Object oriented style

boolean **IntlDateFormatter::setPattern** (string `$pattern`)

Procedural style

boolean **datefmt_set_pattern** (IntlDateFormatter `$fmt`, string `$pattern`)

Set the pattern used for the IntlDateFormatter.

Parameters

`fmt`
The formatter resource.

`pattern`
New pattern string to use.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Bad formatstrings are usually the cause of the failure.

Examples

Example #150 - datefmt_set_pattern() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  ,"MM/dd/yyyy");
echo "pattern of the formatter is : ".datefmt_get_pattern($fmt);
echo "First Formatted output with pattern is ".datefmt_format( $fmt , 0);
datefmt_set_pattern($fmt,'yyyymmdd hh:mm:ss z');
echo "Now pattern of the formatter is : ".datefmt_get_pattern($fmt);
echo "Second Formatted output with pattern is ".datefmt_format( $fmt , 0);
```

```
?>
```

Example #151 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN,"MM/dd/yyyy"  );
echo "pattern of the formatter is : ".$fmt->getPattern();
echo "First Formatted output is ".datefmt_format( $fmt , 0);
$fmt->setPattern('yyyymmdd hh:mm:ss z');
echo "Now pattern of the formatter is : ".$fmt->getPattern();
echo "Second Formatted output is ".datefmt_format( $fmt , 0);

?>
```

The above example will output:

```
pattern of the formatter is : MM/dd/yyyy
First Formatted output with pattern is 12/31/1969
Now pattern of the formatter is : yyyymmdd hh:mm:ss z
Second Formatted output with pattern is 19690031 04:00:00 PST
```

See Also

- datefmt_get_pattern()
- datefmt_create()

IntlDateFormatter::setTimeZoneId

datefmt_set_timezone_id

IntlDateFormatter::setTimeZoneId -- datefmt_set_timezone_id -- Sets the time zone to use

Description

Object oriented style

boolean **IntlDateFormatter::setTimeZoneId** (string `$zone`)

Procedural style

boolean **datefmt_set_timezone_id** (IntlDateFormatter `$fmt`, string `$zone`)

Sets the time zone to use.

Parameters

`fmt`
: The formatter resource.

`zone`
: The time zone ID string of the time zone to use. If **NULL** or the empty string, the default time zone for the runtime is used.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #152 - datefmt_set_timezone_id() example

```
<?php
$fmt = datefmt_create(  "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "timezone_id of the formatter is : ".datefmt_get_timezone_id($fmt);
datefmt_set_timezone_id($fmt,'CN');
echo "Now timezone_id of the formatter is : ".datefmt_get_timezone_id($fmt);
?>
```

Example #153 - OO example

```
<?php
$fmt = new IntlDateFormatter( "en_US"
,IntlDateFormatter::FULL,IntlDateFormatter::FULL,'America/Los_Angeles',IntlD
ateFormatter::GREGORIAN  );
echo "timezone_id of the formatter is : ".$fmt->getTimezoneId();
$fmt->setTimezoneId('CN');
echo "Now timezone_id of the formatter is : ".$fmt->getTimezoneId();

?>
```

The above example will output:

```
timezone_id of the formatter is : America/Los_Angeles
Now timezone_id of the formatter is : CN
```

See Also

- datefmt_get_timezone_id()
- datefmt_create()