Punic Documentation

Release 3.6.2-dev

Michele Locati

Aug 25, 2021

Contents:

1 Installation 3

2 Classes 5

3 Language Data 11

Punic is an easy way to access the Unicode CLDR localization data.

It's a library for PHP 5.3+ that does not have any dependency on other libraries.

CHAPTER 1

Installation

1.1 Manual installation

In case you can't use composer (or even if you don't know it), you can manually add Punic to your project and use it. Simply download the latest version from the Punic releases page.

Extract the zip archive somewhere accessible by your project and make sure you include the `punic.php` file found in the zip archive.

1.2 Installation using composer

As Punic is written using PSR-4, it's highly recommended to use composer to use it. Begin by editing your composer project by running.

```
composer require "punic/punic:^3"
```

That's all it takes, you're ready to go! Have a look at the example on the right to see how a basic call to punic can look like.

CHAPTER 2

Classes

Punic offers a few classes that you can use in your projects.

2.1 `Punic\Calendar` class

The calendar class contains a number of methods to localize date object related outputs. `formatDate` to format a date, `getEraName` to return the name of an era, `getMonthName` to do the same for months. You can also find some helper methods like `toDateTime` to convert a date/time representation to a `DateTime` object.

```
use Punic\Calendar;

include 'vendor/autoload.php';

$dt = Calendar::toDateTime('2010-03-07');

echo Calendar::getEraName(1);
// Output: AD

echo Calendar::getMonthName($dt, 'abbreviated');
// Output: Mar

echo Calendar::getMonthName($dt, 'wide', 'it');
// Output: marzo

echo Calendar::getWeekdayName(1);
// Output: Monday

echo Calendar::getQuarterName(1);
// Output: 1st quarter

echo Calendar::format($dt, 'd MMMM y', 'de');
// Output: 7 März 2010
```

(continues on next page)

(continued from previous page)

```
echo Calendar::formatDate($dt, 'full', 'it');
// Output: domenica 7 marzo 2010

echo Calendar::formatDate($dt, '~yMd', 'da');
// Output: 7/3/2010

$dt2 = Calendar::toDateTime('2010-03-10');
echo Calendar::formatInterval($dt, $dt2, 'yMMMd');
// Output: Mar 7 - 10, 2010
```

2.2 `Punic\Currency` class

Need to know the currency used in a country? Do you want to have a list of all the currencies and their localized name? Use this class!

```
use Punic\Currency;

echo Currency::getCurrencyForTerritory('US');
// Output: USD (the currency code for US Dollars)

echo Currency::getName('USD');
// Output: the name of USD in the current default Punic locale

var_export(Currency::getCurrencyHistoryForTerritory('IT'));
// Output: the history of currencies used in Italy (from Italian Lira to Euro)
```

2.3 `Punic\Data` class

If you work with a larger application, you probably don't want to specify the locale for each string you want to localize.

You can use the `Data` class to set a globally active locale and then skip the locale parameter when calling a method to localize a string.

```
use Punic\Data;
use Punic\Language;

Data::setDefaultLocale('de_DE');
echo Language::getName('de_CH');
// Output: Schweizer Hochdeutsch (Schweiz)
// without specifying a second parameter for Language::getName()

Data::setFallbackLocale('it_IT');
echo Language::getName('de_CH', 'et_INVALID');
// Output: alto tedesco svizzero (Svizzera)
// because the value in the second parameter of Language::getName() isn't a valid␣
↪locale
```

2.4 Punic\Language class

The language class helps you to convert a locale code into a language name. Want to know what it_IT is called in American English? Use this class to show all the languages in the right way.

```
use Punic\Language;

echo Language::getName('it_IT', 'en_US');
// Output: Italian (Italy)

echo Language::getName('de_CH', 'de_DE');
// Output: Schweizer Hochdeutsch (Schweiz)
```

2.5 Punic\Misc class

Use the misc class to join a list of numbers correctly into a single string.

```
use Punic\Misc;

echo Misc::joinUnits(array('1 yd', '1 ft', '3 in'), '', 'en');
// Output: 1 yd, 1 ft, 3 in

echo Misc::joinUnits(array('1 yd', '1 ft', '3 in'), '', 'de');
// Output: 1 yd, 1 ft und 3 in

echo Misc::joinAnd(array('One', 'two', 'three'), '', 'en');
// Output: One, two, and three

echo Misc::joinAnd(array('Eins', 'Zwei', 'Drei'), '', 'de');
// Output: Eins, Zwei und Drei

echo Misc::joinOr(array('One', 'two', 'three'), '', 'en');
// Output: One, two, or three

echo Misc::joinOr(array('Eins', 'Zwei', 'Drei'), '', 'de');
// Output: Eins, Zwei oder Drei
```

2.6 Punic\Number class

The number class offers methods to convert a numerical variable into a properly formatted string. After all, a string like 1,234.56 is easier to read than 1234.56.

```
use Punic\Number;

echo Number::format(1234.567, 2, 'it');
// Output: 1.234,57

echo Number::format(1234.567, 2, 'en');
// Output: 1,234.57

echo Number::unformat('1.234,56', 'it');
// Output: 1234.56
```

(continues on next page)

(continued from previous page)

```
echo Number::unformat('1,234.56', 'en');
// Output: 1234.56
```

2.7 Punic\Phone class

Need to know the country calling codes of a country? Need to know the countries associated to a country calling code? Use this class!

```
use Punic\Phone;

var_export(Phone::getPrefixesForTerritory('US'));
// Output: an array containing '1', the country calling code for US

var_export(Phone::getTerritoriesForPrefix('1'));
// Output: an array containing the country codes having '1' as country calling code
// (for instance US, CA)
```

2.8 Punic\Plural class

Need to know the plural of a number? Use this class.

```
use Punic\Plural;

echo Plural::getRule(1, 'en');
// Output: one

echo Plural::getRule(2, 'en');
// Output: other
```

2.9 Punic\Script class

The Script class contains several methods to work with Scripts (Script is the Unicode term that defines how languages are written, like *Latin* or *Traditional Han*).

```
use Punic\Script;

var_dump(Territory::getAllScriptCodes());
// Output: the identifiers of all the available scripts

var_dump(Territory::getAvailableScriptCodes('en'));
// Output: the identifiers of all the scripts that have available translations in a␣
↪specific language

echo Script::getScriptName('Hant');
// Output: the translated name of the Hant script

var_dump(Territory::getAllScripts());
// Output: an array whose array are the script identifiers and the values are the␣
↪translated script names
```

(continues on next page)

(continued from previous page)

2.10 `Punic\Territory` class

The territory class contains several methods to work with regions, countries and continents.

```
use Punic\Territory;

echo Territory::getName('US', 'en');
// Output: United States

$countries = Territory::getCountries();
print_r($countries);
// Output: the list of all countries, indexed by the territory code

$continents = Territory::getContinents();
print_r($continents);
// Output: the list of all continents

$continentsAndCountries = Territory::getContinentsAndCountries();
print_r($continentsAndCountries);
// Output: the list of all countries grouped by continent
```

2.11 `Punic\Unit` class

Whether you need an abbreviation or a long form of a number with a unit, use this class. Properly formatted numbers and units in every the language.

```
use Punic\Unit;

echo Unit::format(2.0123, 'millisecond', 2, 'it');
// Output: 2,01 ms

echo Unit::format(2.0123, 'millisecond', 'long,1', 'it');
// Output: 2,0 millisecondi

echo Unit::format(2.0123, 'millisecond', 'narrow', 'it');
// Output: 2,0123 ms
```

CHAPTER 3

Language Data

To keep the size of the package at a reasonable level, only data for a limited set of languages (about 40, defined here) is included by default. If you need more, you can get them yourself using the bundled punic-data script.

To get help about how to run this command:

```
./bin/punic-data --help
```

(On Windows, use \ instead of /)

Please remark that when you upgrade your locally installed Punic with Composer, the language files are reverted to the default ones. In the case you want to be sure that you have the languages you need even after Punic is upgraded (for instance, af and zu_ZA), you can add this section to your composer.json file:

```
"scripts": {
    "post-install-cmd": [
        "punic-data --locale=+af,+zu_ZA"
    ],
    "post-update-cmd": [
        "punic-data --locale=+af,+zu_ZA"
    ]
}
```

Language data is fetched from a remote repository. If you plan to install many languages on a regular basis, you can also evaluate to use a local language repository, fetching this repository and instructing `punic-data` to use it with the `--source-location` option.

The default data used by Punic is the one generated by punic/data. You can clone it and use it as the data source.