Navigator Documentation

Release 1.0.0

Simon Holywell

February 18, 2016

Contents

A PHP library for geographic calculations:

- Calculate the distance between two coordinate points on the earth's surface (using Vincenty, Haversine, Great Circle or The Cosine Law)
- Conversion between units (metres to kilometres, nautical miles and miles).
- Convert coordinate notation (decimals to degrees, minutes & seconds and back again).

This is an improved (PHP5.3.2+) and tested version of Geographic Calculations in PHP.

Contents:

CHAPTER 1

Installation

There are a few ways of installing this library, the easiest of which is via Packagist with Composer.

1.1 Packagist with Composer

Composer is a great way to manage dependencies for PHP projects and there is a ready made package for Navigator available on Packagist.

In your projects *composer.json* file you should enter the following information:

```
{
    "require": {
        "treffynnon/navigator": "1.*"
    }
}
```

Next you need to install composer on your system with

```
curl -s http://getcomposer.org/installer | php
```

Install the Composer managed dependencies with:

```
php composer.phar install
```

You will now have a *vendors* directory in your project that contains Navigator.

Composer also automatically generates an autoload file that you can use to autoload the classes of the Navigator library (and any other dependencies you have managed with Composer). To do this add the following to your projects bootstrap file:

```
<?php require 'vendor/autoload.php';
```

Navigator is now installed in your project with Composer meaning that it is easy to keep up to date!

1.2 git Clone or Zip Package

Navigator can also be installed from source by either using git to clone or export the repository or by downloading a zipped release from GitHub.

To obtain the library with git it is as simple as:

```
git clone git://github.com/treffynnon/Navigator.git
```

Otherwise you can download a zip or tar file of the latest release from GitHub and extract it.

Then to initialise the autoloader for the Navigator library add the following to your projects bootstrap:

```
<?php
require_once __DIR__ . 'Navigator/lib/Treffynnon/Navigator.php';
use Treffynnon\Navigator as N;
N::autoloader();
```

The Navigator library is now available to your project.

CHAPTER 2

Quickstart Tutorial

Please ensure you have completed the installation instructions for the Navigator library before continuing with these quickstart tutorials.

2.1 Super Simple Example

This is the easiest way to get a quick distance between two points of the Earth in metres.

```
<?php
use Treffynnon\Navigator as N;
$distance = N::getDistance(10, 81.098, 15.6, '5° 10\' 11.009"W');
```

The function takes a sequence of latitude and longitude values:

N::getDistance(lat1, long1, lat2, long2)

Returns the distance in metres between the supplied points on Earth

Parameters

- **lat1** (*string or float*) – The latitude of point 1
- **long1** (*string or float*) – The longitude of point 1
- **lat2** (*string or float*) – The latitude of point 2
- **long2** (*string or float*) – The longitude of point 2

Return type float

2.2 A Slightly More Advanced Example

To get more control over the setup of the distance calculation you can make use of the distance factory. The following snippet will give the *$distance* using the Haversine formula and converted to parsecs.

```
<?php
use Treffynnon\Navigator as N;
use Treffynnon\Navigator\Distance\Calculator\Haversine as H;
use Treffynnon\Navigator\Distance\Converter\MetreToParsec as P;
$Distance = N::distanceFactory(10, 81.098, 15.6, '5° 10\' 11.009"W');
$distance = $Distance->get(new H, new P);
```

N::distanceFactory(lat1, long1, lat2, long2)

Get a distance instance pre-populated with the supplied sequence of latitude and longitude values

Parameters

- **lat1** (*string or float*) – The latitude of point 1
- **long1** (*string or float*) – The longitude of point 1
- **lat2** (*string or float*) – The latitude of point 2
- **long2** (*string or float*) – The longitude of point 2

Return type TreffynnonNavigatorDistance

CHAPTER 3

Coordinates

The coordinate class must be combined with LatLong to create a point on the celestial bodies surfaces (most commonly this is the Earth). It handles the storage of a supplied coordinate value and its conversion to radians for internal use by Calculators.

This scheme makes it easy to supply a custom coordinate parser or specify whether to use *Decimal* or *Degrees Minutes Seconds* notation from the standard set of parsers.

```
<?php
use Treffynnon\Navigator\Coordinate as C;
$coord = new C('5° 10\' 11.009"W', new C\DmsParser);
```

3.1 Custom Parser

Creating a custom parser is as simple as extending *Treffynnon\Navigator\Coordinate\ParserAbstract* like in this Radian parsing example

```
<?php
namespace YourProject\Navigator\Coordinate;
use Treffynnon\Navigator\Coordinate as C;
class RadianParser extends C\ParserAbstract {
    public function parse($coord) {
        return $coord;
    }
    public function get($coord) {
        return $coord;
    }
}
```

Then you can put it into action by injecting it into a coordinate instance

```
<?php
use YourProject\Navigator\Coordinate as YPC;
use Treffynnon\Navigator\Coordinate as C;
$coord = new C(1.2175876579, new YPC\RadianParser);
```

Please note the namespace *YourProject\Navigator\Coordinate* should be changed to reflect the real names in your project.

CHAPTER 4

LatLong

It is just a simple construct to combine coordinate instances into a latitude and longitude point. It will also prime the coordinate with its direction (either latitude or longitude). This can later be used by a parser to add in any meta information about a coordinate. This can be most easily seen the in the *get()* method of the DmsParser class.

CHAPTER 5

Distance

When *Distance* is supplied with two instances of *LatLng* it can be used to calculate the distance between the points. It does this by using a *Calculator* such as *Great Circle* and optionally a unit converter such as *MetreToNauticalMile*:

```
<?php
use Treffynnon\Navigator as N;
$coord1 = new N\LatLong(
    new N\Coordinate(10.9978),
    new N\Coordinate(35.6234)
);
$coord2 = new N\LatLong(
    new N\Coordinate(25),
    new N\Coordinate(-13.456)
);
$Distance = new N\Distance($coord1, $coord2);
```

Specify the calculator and conversion on the *get()* method of *Distance*:

```
<?php
use Treffynnon\Navigator\Distance as D;
$distance = $Distance->get(new D\Calculator\GreatCircle,
                           new D\Converter\MetreToNauticalMile);
```

$distance now has the distance value calculated by Great Circle in Nautical Miles.

5.1 Calculators

The Navigator library comes with four distance calculators by default:

- The Cosine Law
- Great Circle
- Haversine
- Vincenty

Of the selection Vincenty is the most accurate and also the default. It is the most computationally intensive, but not prohibitively so by any stretch.

5.1.1 Celestial Bodies

Most commonly and by default Navigator will be using Earth, but it can be altered by passing in a different *Celestial Body* such as *Mars* or the *Moon*:

```
<?php
use Treffynnon\Navigator\CelestialBody\Mars as M;
use Treffynnon\Navigator\Distance as D;
$distance = $Distance->get(new D\Calculator\GreatCircle(new M),
                           new D\Converter\MetreToNauticalMile);
```

Custom Celestial Bodies

Custom celestial bodies are very simple to setup with a set of statistics - see *Treffynnon\Navigator\CelestialBody\CelestialBodyAbstract* for more information.

5.1.2 Custom Calculators

As with coordinate parsers it is a trivial matter to create custom calculators. Simply extend the abstract class - *Treffynnon\Navigator\Distance\Calculator\CalculatorAbstract*.

5.2 Converters

Converters can be used independently of the Navigator library or injected into the *Distance->get()* method. By default Navigator returns distances in metres, but this can be converted to the following units:

- Furlong
- Kilometre
- League
- Mile
- Nautical Mile
- Parsec

An example follows:

```
<?php
use Treffynnon\Navigator\Distance\Converter\MetreToFurlong as F;
$distance = $Distance->get(null, new F);
```

5.2.1 Custom Converters

As with custom calculators, but even simpler! See *Treffynnon\Navigator\Distance\Converter\ConverterAbstract*.

CHAPTER 6

Tests

Tests are written for PHPUnit 3.5+ with 100% code coverage and can be run with:

```
phpunit --bootstrap tests/bootstrap.php tests
```

6.1 Travis-CI

Continuous integration is handled by Travis-CI:

6.2 Code Coverage

Code coverage can be obtained from PHPUnit with the following command:

```
phpunit --bootstrap tests/bootstrap.php --coverage-html ../coverage tests
```

CHAPTER 7

Contribution

Contributions are welcome through pull requests, but they *must* not break any tests and all new features should come with 100% code coverage.

CHAPTER 8

Licence

8.1 BSD 2-Clause License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

CHAPTER 9

Indices and tables

- genindex
- search