

The Geographic Distance and Radius Library is an object oriented class library that provides methods to calculate the distances between latitude and longitude coordinate pairs. The library also includes optimized methods to determine radius inclusion of any distance from a common coordinate.

The library is available in the Perl, PHP, and C++ programming languages. All versions share a common definition and naming convention for the coordinate object therefore we use a single description for each method.

The Perl version is designed for and requires Perl version 5 or above. It includes one source file, *distance.pm*. This is a simple Perl module that requires no specific installation. It may be used with any script by placing it in the same directory as your Perl script.

The PHP version is designed for and requires PHP version 4 or above. It includes one source file, *distance.class.php*. To use the object simply include the class file in your particular script and create the object.

The C++ version includes two files, *distance.h* and *distance.cpp*. This version is delivered as source only. You must include and compile it with your particular application. It uses no compiler specific values or setting and may be compiled with virtually any C++ compiler on any platform.

#### Coordinate Set/Get related methods

```
void Set(double lat,double lon);
int SetLatitude(double deg);
int SetLatitude(double deg,double min = 0.0,double sec = 0.0,int north = TRUE);
int SetLongitude(double deg);
int SetLongitude(double deg,double min = 0.0,double sec = 0.0,int east = TRUE);
double GetLatitude();
double GetLongitude();
char *GetDMSLatitudeString();
char *GetDMSLongitudeString();
```

#### Distance calculations

```
double Distance(double lat,double lon);
double HaversineDistance(double lat,double lon);
double PolarDistance(double lat,double lon);
double DistanceNS(double lat,int asdistance = TRUE);
double DistanceEW(double lon,int asdistance = TRUE);
```

#### Direction related methods

```
double Bearing(double lat,double lon);
void *Direction(double lat,double lon,int asindex = FALSE);
```

#### Radius related methods

```
double inRadius(double lat,double lon,double miles = 0);
void SetRadius(double miles);
double GetRadius();
double GetMinLatitude();
double GetMinLongitude();
double GetMaxLatitude();
double GetMaxLongitude();
```

## ::Coordinate(lat = 0.0, lon = 0.0)

This method is used to instantiate and initialize the coordinate object. You can instantiate or create the object and set its initial values by specifying coordinate values through the lat and lon parameters or optionally create the object and pass no parameters to initialize its coordinates to zero.

Note: For the Perl version only, the object is instantiated by creating a new instance of the distance module which equates to the PHP and C++ Coordinate object.

### Parameters:

*lat*

Optional, specifies latitude coordinate in decimal degree notation. Uses default value of 0.0 if no value is specified.

*lon*

Optional, specifies longitude coordinate in decimal degree notation. Uses default value of 0.0 if no value is specified.

### Examples:

*Perl*

```
# we assume the module distance.pm exists in the current directory
use distance;

my $coord = new distance($lat,$lon);
```

*PHP*

```
<?php
require_once("./distance.class.php");

$coord =& new Coordinate($lat,$lon);
?>
```

*C++*

```
#include "distance.h"
double lat;
double lon;

Coordinate coord1(lat, lon);           // static instantiation
Coordinate *coord = new Coordinate(lat, lon); // dynamic instantiation
```

## ::Distance(lat, lon)

Calculate the great circle distance in statute miles between the current object's coordinates and the specified latitude, longitude coordinate pair. This method uses standard spherical geometry for the calculation and is the default distance calculation used by the object.

### Parameters:

*lat*  
Destination latitude coordinate in decimal degree notation.

*lon*  
Destination longitude coordinate in decimal degree notation.

### Returns:

Distance in statute miles.

### Examples:

#### Perl

```
use distance;
my $coord = new distance($lat,$lon);

# calculate the distance to this set of coordinates
# and print the result

print $coord->Distance($lat2,$lon2);
```

#### PHP

```
<?php
require_once("./distance.class.php");
$coord =& new Coordinate($lat,$lon);

echo $coord->Distance($lat2,$lon2);
?>
```

#### C++

```
#include "distance.h"
double lat;
double lon;
double lat2;
double lon2;

Coordinate coord1(lat, lon);           // static instantiation
Coordinate *coord = new Coordinate(lat, lon); // dynamic instantiation

printf("%f", coord->Distance(lat2,lon2));
```

## ::HaversineDistance(lat, lon)

Calculate the great circle distance in statute miles between the current object's coordinates and the specified latitude, longitude coordinate pair. This method is similar to the standard Distance method but uses the Haversine distance formula yielding greater accuracy for smaller distances between the coordinate pairs.

### Parameters:

*lat*  
Destination latitude coordinate in decimal degree notation.

*lon*  
Destination longitude coordinate in decimal degree notation.

### Returns:

Distance in statute miles.

### Examples:

#### Perl

```
use distance;  
my $coord = new distance($lat,$lon);  
  
print $coord->HaversineDistance($lat2,$lon2);
```

#### PHP

```
<?php  
require_once("./distance.class.php");  
$coord =& new Coordinate($lat,$lon);  
  
echo $coord->HaversineDistance($lat2,$lon2);  
?>
```

#### C++

```
#include "distance.h"  
double lat;  
double lon;  
double lat2;  
double lon2;  
  
Coordinate coord1(lat, lon);           // static instantiation  
Coordinate *coord = new Coordinate(lat, lon); // dynamic instantiation  
  
printf("%f", coord->HaversineDistance(lat2,lon2));
```

## ::PolarDistance(lat, lon)

Calculate the great circle distance in statute miles between the current object's coordinates and the specified latitude, longitude coordinate pair. This method is similar to the standard Distance method but uses the Polar distance flat earth model formula yielding greater accuracy when the coordinate pairs approach polar latitudes.

### Parameters:

*lat*  
Destination latitude coordinate in decimal degree notation.

*lon*  
Destination longitude coordinate in decimal degree notation.

### Returns:

Distance in statute miles.

### Examples:

#### Perl

```
use distance;
my $coord = new distance($lat,$lon);

print $coord->PolarDistance($lat2,$lon2);
```

#### PHP

```
<?php
require_once("./distance.class.php");
$coord =& new Coordinate($lat,$lon);

echo $coord-> PolarDistance ($lat2,$lon2);
?>
```

#### C++

```
#include "distance.h"
double lat;
double lon;
double lat2;
double lon2;

Coordinate coord1(lat, lon);           // static instantiation
Coordinate *coord = new Coordinate(lat, lon); // dynamic instantiation

printf("%f", coord-> PolarDistance (lat2,lon2));
```

## ::DistanceNS(lat, asdistance = TRUE)

Calculate the latitudinal (north/south) distance between the current object's latitude and the specified latitude.

### Parameters:

*lat*

Destination latitude coordinate in decimal degree notation.

*asdistance*

Optional, determines the returned distance type.

TRUE (default value), return value specifies distance in statute miles

FALSE, return value specifies angular distance in radians.

### Returns:

Distance in statute miles.

::DistanceEW(lon, asdistance = TRUE)

Calculate the longitudinal (east/west) distance between the current object's longitude and the specified longitude.

**Parameters:**

*lat*

Destination longitude coordinate in decimal degree notation.

*asdistance*

Optional, determines the returned distance type.

TRUE (default value), return value specifies distance in statute miles

FALSE, return value specifies angular distance in radians.

**Returns:**

Distance in statute miles.

## ::Bearing(lat, lon)

Calculate the compass bearing from the current object's coordinates traveling toward the specified latitude, longitude coordinate pair.

### Parameters:

*lat*  
Destination latitude coordinate in decimal degree notation.

*lon*  
Destination longitude coordinate in decimal degree notation.

### Returns:

Compass bearing as a decimal degree.

## ::Direction(lat, lon, asindex = FALSE)

Return the generalized compass direction from the current object's coordinates traveling toward the specified latitude, longitude coordinate pair.

### Parameters:

*lat*  
Destination latitude coordinate in decimal degree notation.

*lon*  
Destination longitude coordinate in decimal degree notation.

*asindex*  
Optional, determines the returned direction type.  
FALSE (default value), result is returned as a textual description  
TRUE, the result is returned as an index

### Returns:

asindex equals FALSE, descriptive string, North, South, etc.

asindex equals TRUE,

- 0 = North
- 1 = North East
- 2 = East
- 3 = South East
- 4 = South
- 5 = South West
- 6 = West
- 7 = North West

## ::inRadius(lat, lon, miles = 0)

Determine if the specified latitude, longitude coordinate pair fall within a specified radius from the object's current coordinates. This method is generally preceded by a preinitialization call to the SetRadius method but may achieve the identical results by passing the optional miles parameter.

### Parameters:

*lat*

Destination latitude coordinate in decimal degree notation.

*lon*

Destination longitude coordinate in decimal degree notation.

*miles*

Optional, specifies or resets the object's radius in miles used for inclusion test.

### Returns:

Distance in statute miles if the specified coordinate falls within the current radius from the current object's coordinates else -1 if the coordinate is outside the radius.

## ::SetRadius(miles)

This method is used to initialize the object for optimized subsequent radius inclusion testing.

### Parameters:

*miles*

Specifies or resets the object's radius in miles used for inclusion test.

## ::GetRadius()

Return the current radius in statute miles used for radius inclusion testing.

## ::GetMinLatitude()

Return the current minimum latitude accepted for radius inclusion.

## ::GetMinLongitude()

Return the current minimum longitude accepted for radius inclusion.

## ::GetMaxLatitude()

Return the current maximum latitude accepted for radius inclusion.

## ::GetMaxLongitude()

Return the current maximum longitude accepted for radius inclusion.

**Examples:**

*Perl*

```
use distance;

my $coord = new distance(28.8619,-81.6189);
$coord->SetRadius(25) ;

# create our coordinate object and initialize a 25 mile radius

$coord = new distance(28.8619,-81.6189);
$coord->SetRadius(25);

# build the SQL query to select all coordinates that fall in to
# the rectangular bounds established by our call to SetRadius

$query = 'SELECT * FROM database WHERE';
$query .= 'Latitude <= '.$coord->GetMaxLatitude();
$query .= 'AND Latitude >= '.$coord->GetMinLatitude();
$query .= 'AND Longitude <= '.$coord->GetMaxLongitude();
$query .= 'AND Longitude >= '.$coord->GetMinLongitude();

# execute the query ... we assume success ... the SQL engine will
# eliminate all coordinates that are clearly outside our bounds ...

$db = $dbh->prepare($query);
$db->execute;

# now loop through the query results for final radius inclusion testing
# if a point falls within our radius then print them out along with
# the distance from our center coordinate and the direction

while (@tmp = $db->fetchrow_array)
{
    ($lat,$lon) = @tmp;

    if (($dist = $coord->inRadius($lat,$lon)) > 0)
        {print sprintf("%s %s %3.3f miles %s<br>",$lat,$lon,$dist,$coord->Direction($lat,$lon));}
}
```

*PHP*

```
<?php
require_once("./distance.class.php");

// create the object and set inclusion radius to 25 miles
$coord =& new Coordinate(28.8619,-81.6189);
$coord->SetRadius(25) ;

// now lets do a simple brute force sequential search through our
// database to find all coordinates within our radius

$handle = fopen (".datamaster.csv", "r");
while ($data = fgetcsv ($handle, 1000, ","))
{
    if (($dist = $coord->inRadius($data[0],$data[1])) > 0)
        print sprintf("%s %s %3.3f miles %s<br>",
            $data[0],$data[1],$dist,$coord->Direction($data[0],$data[1]));
}

fclose ($handle);
?>
```

## ::Set(double lat,double lon)

Initialize (set) the current object's latitude and longitude coordinate values for subsequent operations.

### Parameters:

*lat*  
Latitude coordinate in decimal degree notation.

*lon*  
Longitude coordinate in decimal degree notation.

## ::SetLatitude(deg, min = 0.0, sec = 0.0, north = TRUE)

## ::SetLongitude(deg, min = 0.0, sec = 0.0, east = TRUE)

Initialize (set) the current object's latitude and longitude coordinate values for subsequent operations using either decimal degree or DMS notational values.

### Parameters:

*deg*  
specifies the coordinate value as a decimal degree or the whole degree value if using DMS notation.

*min*  
specifies the coordinate minutes value when using DMS notation

*sec*  
specifies the coordinate seconds value when using DMS notation

*north/east*  
when using DMS notation identifies the coordinate value as north/east or south/west

## ::GetLatitude()

Return the current latitude coordinate of the object as a decimal degree.

## ::GetLongitude()

Return the current longitude coordinate of the object as a decimal degree.

## ::GetDMSLatitudeString()

Return the current latitude coordinate of the object as a DMS formatted string.

## ::GetDMSLongitudeString()

Return the current longitude coordinate of the object as a DMS formatted string.