



[Main Page](#)
[The map](#)
[Mapping projects](#)
[Map Features](#)
[Contributors](#)
[Help](#)
[Blog](#)
[Shop](#)
[Donations](#)
[Recent changes](#)

▼ [Toolbox](#)
[What links here](#)
[Related changes](#)
[Special pages](#)
[Printable version](#)
[Permanent link](#)
[Cite this page](#)

Page [Discussion](#)

Read

[View source](#)
[View history](#)



[Find out more about OpenStreetMap's upcoming license change \(translations\) \(discussion\)](#)

Slippy map tilenames

This article describes the **file naming conventions for the [Slippy Map](#) application**.

- Tiles are 256 × 256 pixel PNG files
- Each zoom level is a directory, each column is a subdirectory, and each tile in that column is a file
- Filename(url) format is /zoom/x/y.png


The slippy map expects tiles to be served up at URLs following this scheme, so all tile server URLs look pretty similar.

Contents [\[hide\]](#)


- [1 Tile servers](#)
- [2 Zoom levels](#)
- [3 X and Y](#)
- [4 Derivation of tile names](#)
- [5 Implementations](#)
 - [5.1 Pseudo-Code](#)
 - [5.1.1 lon/lat to tile numbers](#)
 - [5.1.2 tile numbers to lon/lat](#)
 - [5.2 Math matics](#)
 - [5.3 Python](#)
 - [5.3.1 lon/lat to tile numbers](#)
 - [5.3.2 tile numbers to lon/lat](#)
 - [5.4 Perl](#)
 - [5.4.1 lon/lat to tile numbers](#)
 - [5.4.2 tile numbers to lon/lat](#)
 - [5.5 PHP](#)
 - [5.5.1 lon/lat to tile numbers](#)
 - [5.5.2 tile numbers to lon/lat](#)
 - [5.6 ECMAScript \(JavaScript/ActionScript etc.\)](#)
 - [5.7 C/C++](#)
 - [5.8 Java](#)
 - [5.8.1 compute bounding box for tile number](#)
 - [5.9 VB.Net](#)
 - [5.10 C#](#)
 - [5.11 XSLT](#)
 - [5.12 Scala](#)
 - [5.13 Revolution/Transcript](#)
 - [5.14 Mathematica](#)
 - [5.15 Tcl](#)
 - [5.15.1 lat/lon to tile number](#)

- 5.15.2 tile number to lat/lon
- 5.16 Pascal
 - 5.16.1 coordinates to tile numbers
 - 5.16.2 tile numbers to coordinates
- 5.17 R
 - 5.17.1 coordinates to tile numbers
- 6 Subtiles
- 7 Tools
- 8 References

Tile servers



It has been proposed that this page or section be merged with [TMS](#). [\(Discuss\)](#)





The first part of the URL specifies the tile server, and perhaps other parameters which might influence the style.

Generally several subdomains (server names) are provided to get around browser limitations on the number of simultaneous HTTP connections to each host. Browser-based applications can thus request multiple tiles from multiple subdomains faster than from one subdomain. For example, OSM, [OpenCycleMap](#) and [CloudMade](#) servers have three subdomains (a.tile, b.tile, c.tile), [MapQuest](#) has four (otile1, otile2, otile3, otile4), all pointing to the same CDN.

That all comes before the /zoom/x/y .png tail.

Here are some examples:

Name	URL template	zoomlevels
OSM Mapnik	<code>http://[abc].tile.openstreetmap.org/zoom/x/y.png</code>	0-18
OSM Osmarender/Tiles@Home	<code>http://[abc].tah.openstreetmap.org/Tiles/tile/zoom/x/y.png</code>	0-17
OpenCycleMap	<code>http://[abc].tile.opencyclemap.org/cycle/zoom/x/y.png</code>	0-16
OpenCycleMap Transport (<i>experimental</i>)	<code>http://[abc].tile2.opencyclemap.org/transport/zoom/x/y.png</code>	0-18
CloudMade (Web style)	<code>http://[abc].tile.cloudmade.com/your_CloudMade_API_key/1/256/zoom/x/y.png</code>	0-18
CloudMade (Fine line style)	<code>http://[abc].tile.cloudmade.com/your_CloudMade_API_key/2/256/zoom/x/y.png</code>	0-18
CloudMade (NoNames style)	<code>http://[abc].tile.cloudmade.com/your_CloudMade_API_key/3/256/zoom/x/y.png</code>	0-18
MapQuest	<code>http://otile[1234].mqcdn.com/tiles/1.0.0/osm/zoom/x/y.png</code>	0-18
MapQuest Open Aerial	<code>http://oatile[1234].mqcdn.com/naip/zoom/x/y.png</code>	0-11 globally, 12+ in the United States 
Migurski's Terrain 	<code>http://tile.stamen.com/terrain-background/zoom/x/y.jpg</code>	4-18, US-only (for now)

Further tilesets are available from various '3rd party' sources.

Zoom levels

The zoom parameter is an integer between 0 (zoomed out) and 18 (zoomed in). 18 is normally the maximum, but some tile servers might go beyond that.

0	1 tile covers whole world	1 tile
1	2 × 2 tiles	4 tiles

2	4 × 4 tiles	16 tiles
n	$2^n \times 2^n$ tiles	2^{2n} tiles
12	4096 × 4096 tiles	16.777.216
16	Maximum zoom for OpenCycleMap (mostly)	$2^{32} = 4.294.967.296$ tiles
17	Maximum zoom for Osmarender layer	17.179.869.184 tiles
18	Maximum zoom for Mapnik layer	68.719.476.736 tiles

X and Y

- X goes from 0 (left edge is 180 °W) to $2^{2^{zoom}} - 1$ (right edge is 180 °E)
- Y goes from 0 (top edge is 85.0511 °N) to $2^{2^{zoom}} - 1$ (bottom edge is 85.0511 °S) **in a Mercator projection**

For the curious, the num 85.0511 is the result of $\arctan(\sinh(\pi))$. By using this bound, the entire map becomes a (very large) square. See also the [Osmarender bug](#).

Derivation of tile names

- Reproject the coordinates to the Mercator projection:
 - $x = lon$
 - $y = \log(\tan(lat) + \sec(lat))$
(lat and lon are in radians)
- Transform range of x and y to 0 - 1 and shift origin to top left corner:
 - $x = (1 + (x / \pi)) / 2$
 - $y = (1 - (y / \pi)) / 2$
- Calculate the number of tiles across the map, n , using $2^{2^{zoom}}$
- Multiply x and y by n . Round results down to give *tilex* and *tiley*.

Implementations

Pseudo-Code

For those who like pseudo-code, here's some hints: ($\sec == 1/\cos$) Please note that \log stands for logarithmus naturalis (also known as $\ln(x)$), not decimals, as used on some calculators.

lon/lat to tile numbers

```
n = 2 ^ zoom
xtile = ((lon_deg + 180) / 360) * n
ytile = (1 - (log(tan(lat_rad) + sec(lat_rad)) / pi)) / 2 * n
```

tile numbers to lon/lat

```
n = 2 ^ zoom
lon_deg = xtile / n * 360.0 - 180.0
lat_rad = arctan(sinh(pi * (1 - 2 * ytile / n)))
```

```
lat_deg = lat_rad * 180.0 / π
```

Mathematics

Idem with mathématique signs

$$\left\lfloor \frac{x_{tile}}{2^{zoom}} \right\rfloor$$
$$\left\lfloor \frac{\ln \left(\frac{\tan \left(\frac{lat}{180} * \pi \right) + \frac{1}{\cos \left(\frac{lat}{180} * \pi \right)}}{\pi} \right)}{2^{(zoom-1)}} \right\rfloor$$

$$lon = (x_{tile} / 2^{zoom} * 360) - 180$$

$$lat = \arctan(\sinh((\pi * [1 - (2 * y_{tile} / 2^{zoom})])) * 180 / \pi)$$

$\lfloor \rfloor$ Partie entiere de la valeur
lat = latitude ; lon = longitude

Python

lon/lat to tile numbers

```
import math
def deg2num(lat_deg, lon_deg, zoom):
    lat_rad = math.radians(lat_deg)
    n = 2.0 ** zoom
    xtile = int((lon_deg + 180.0) / 360.0 * n)
    ytile = int((1.0 - math.log(math.tan(lat_rad) + (1 / math.cos(lat_rad)))) / math.pi / 2.0 * n)
    return (xtile, ytile)
```

tile numbers to lon/lat

```
import math
def num2deg(xtile, ytile, zoom):
    n = 2.0 ** zoom
    lon_deg = xtile / n * 360.0 - 180.0
    lat_rad = math.atan(math.sinh(math.pi * (1 - 2 * ytile / n)))
    lat_deg = math.degrees(lat_rad)
    return (lat_deg, lon_deg)
```

This returns the NW-corner of the square. Use the function with xtile+1 and/or ytile+1 to get the other corners. With xtile+0.5 & ytile+0.5 it will return the center of the tile.

[Another python implementation](#)

Perl

lon/lat to tile numbers

```
use Math::Trig;
sub getTileNumber {
    my ($lat,$lon,$zoom) = @_;
    my $xtile = int( ($lon+180)/360 *2**$zoom );
    my $ytile = int( (1 - log(tan(deg2rad($lat)) + sec(deg2rad($lat)))/pi)/2 *2**$zoom );
```

```

    return ($xtile, $ytile);
}

```

tile numbers to lon/lat

```

use Math::Trig;
sub Project {
    my ($X,$Y, $Zoom) = @_ ;
    my $Unit = 1 / (2 ** $Zoom);
    my $rely1 = $Y * $Unit;
    my $rely2 = $rely1 + $Unit;

    # note: $LimitY = ProjectF(degrees(atan(sinh(pi)))) = log(sinh(pi)+cosh(pi)) = pi
    # note: degrees(atan(sinh(pi))) = 85.051128..
    #my $LimitY = ProjectF(85.0511);

    # so stay simple and more accurate
    my $LimitY = pi;
    my $RangeY = 2 * $LimitY;
    $rely1 = $LimitY - $RangeY * $rely1;
    $rely2 = $LimitY - $RangeY * $rely2;
    my $Lat1 = ProjectMercToLat($rely1);
    my $Lat2 = ProjectMercToLat($rely2);
    $Unit = 360 / (2 ** $Zoom);
    my $Long1 = -180 + $X * $Unit;
    return ($Lat2, $Long1, $Lat1, $Long1 + $Unit); # S,W,N,E
}
sub ProjectMercToLat($){
    my $Mercy = shift;
    return rad2deg(atan(sinh($Mercy)));
}
sub ProjectF
{
    my $Lat = shift;
    $Lat = deg2rad($Lat);
    my $Y = log(tan($Lat) + sec($Lat));
    return $Y;
}

```

PHP

lon/lat to tile numbers

```

$xtile = floor((( $lon + 180) / 360) * pow(2, $zoom));
$ytile = floor((1 - log(tan(deg2rad($lat))) + 1 / cos(deg2rad($lat))) / pi()) / 2 * pow(2, $zoom));

```

tile numbers to lon/lat

```

$n = pow(2, $zoom);
$lon_deg = $xtile / $n * 360.0 - 180.0;
$lat_deg = rad2deg(atan(sinh(pi() * (1 - 2 * $ytile / $n))));

```

ECMAScript (JavaScript/ActionScript etc.)

```

function long2tile(lon,zoom) { return (Math.floor((lon+180)/360*Math.pow(2,zoom))); }
function lat2tile(lat,zoom) { return (Math.floor((1-Math.log(Math.tan(lat*Math.PI/180) + 1/Math.cos(lat*Math.PI/180))/Math.PI)/2 *Math.pow(2,zoom))); }

```

Inverse process:

```
function tile2long(x,z) {
  return (x/Math.pow(2,z)*360-180);
}
function tile2lat(y,z) {
  var n=Math.PI-2*Math.PI*y/Math.pow(2,z);
  return (180/Math.PI*Math.atan(0.5*(Math.exp(n)-Math.exp(-n))));
}
```

Example: [Tilesname WebCalc V1.0](#)

C/C++

```
int long2tilex(double lon, int z)
{
    return (int)(floor((lon + 180.0) / 360.0 * pow(2.0, z)));
}

int lat2tiley(double lat, int z)
{
    return (int)(floor((1.0 - log( tan(lat * M_PI/180.0) + 1.0 / cos(lat * M_PI/180.0)) / M_PI) / 2.0 * pow(2.0, z)));
}

double tilex2long(int x, int z)
{
    return x / pow(2.0, z) * 360.0 - 180;
}

double tiley2lat(int y, int z)
{
    double n = M_PI - 2.0 * M_PI * y / pow(2.0, z);
    return 180.0 / M_PI * atan(0.5 * (exp(n) - exp(-n)));
}
```

Java

```
public class slippytest {
  public static void main(String[] args) {
    int zoom = 10;
    double lat = 47.968056d;
    double lon = 7.909167d;
    System.out.println("http://tile.openstreetmap.org/" + getTileNumber(lat, lon, zoom) + ".png");
  }
  public static String getTileNumber(final double lat, final double lon, final int zoom) {
    int xtile = (int)Math.floor( (lon + 180) / 360 * (1<<zoom) );
    int ytile = (int)Math.floor( (1 - Math.log(Math.tan(Math.toRadians(lat))) + 1 / Math.cos(Math.toRadians(lat))) / Math.PI) / 2 * (1<<zoom) );
    return (" " + zoom + "/" + xtile + "/" + ytile);
  }
}
```

compute bounding box for tile number

```
class BoundingBox {
  double north;
  double south;
  double east;
  double west;
}

BoundingBox tile2boundingBox(final int x, final int y, final int zoom) {
  BoundingBox bb = new BoundingBox();
```

```

        bb.north = tile2lat(y, zoom);
        bb.south = tile2lat(y + 1, zoom);
        bb.west = tile2lon(x, zoom);
        bb.east = tile2lon(x + 1, zoom);
        return bb;
    }

    static double tile2lon(int x, int z) {
        return x / Math.pow(2.0, z) * 360.0 - 180;
    }

    static double tile2lat(int y, int z) {
        double n = Math.PI - (2.0 * Math.PI * y) / Math.pow(2.0, z);
        return Math.toDegrees(Math.atan(Math.sinh(n)));
    }
}

```

VB.Net

```

Private Function CalcTileXY(ByVal lat As Single, ByVal lon As Single, ByVal zoom As Long) As Point
    CalcTileXY.X = CLng(Math.Floor((lon + 180) / 360 * 2 ^ zoom))
    CalcTileXY.Y = CLng(Math.Floor((1 - Math.Log(Math.Tan(lat * Math.PI / 180) + 1 / Math.Cos(lat * Math.PI / 180)) / Math.PI) / 2 * 2 ^ zoom))
End Function

```

C#

```

public PointF WorldToTilePos(double lon, double lat, int zoom)
{
    PointF p = new Point();
    p.X = (float)((lon + 180.0) / 360.0 * (1 << zoom));
    p.Y = (float)((1.0 - Math.Log(Math.Tan(lat * Math.PI / 180.0) +
        1.0 / Math.Cos(lat * Math.PI / 180.0)) / Math.PI) / 2.0 * (1 << zoom));

    return p;
}

public PointF TileToWorldPos(double tile_x, double tile_y, int zoom)
{
    PointF p = new Point();
    double n = Math.PI - ((2.0 * Math.PI * tile_y) / Math.Pow(2.0, zoom));

    p.X = (float)((tile_x / Math.Pow(2.0, zoom) * 360.0) - 180.0);
    p.Y = (float)(180.0 / Math.PI * Math.Atan(Math.Sinh(n)));

    return p;
}

```

XSLT

Requires math extensions from exslt.org.

```

<xsl:transform
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:m="http://exslt.org/math"
  extension-element-prefixes="m"
  version="1.0">

  <xsl:output method="text"/>
  <xsl:variable name="pi" select="3.14159265358979323846"/>

```

```

<xsl:template name="tile">
  <xsl:param name="lat"/>
  <xsl:param name="zoomfact"/>
  <xsl:variable name="a" select="($lat * $pi) div 180.0"/>
  <xsl:variable name="b" select="m:log(m:tan($a) + (1.0 div m:cos($a)))"/>
  <xsl:variable name="c" select="(1.0 - ($b div $pi)) div 2.0"/>
  <xsl:value-of select="floor($c * $zoomfact)"/>
</xsl:template>

<xsl:template name="tilename">
  <xsl:param name="lat"/>
  <xsl:param name="lon"/>
  <xsl:param name="zoom" select="10"/>
  <xsl:variable name="zoomfact" select="m:power(2,$zoom)"/>
  <xsl:variable name="x" select="floor((360.0 + ($lon * 2)) * $zoomfact div 720.0)"/>
  <xsl:variable name="y">
    <xsl:call-template name="tile">
      <xsl:with-param name="lat" select="$lat"/>
      <xsl:with-param name="zoomfact" select="$zoomfact"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:value-of select="concat($zoom, '/', $x, '/', $y)"/>
</xsl:template>

<xsl:template match="/">
  <xsl:call-template name="tilename">
    <xsl:with-param name="lat" select="49.867731999999997"/>
    <xsl:with-param name="lon" select="8.6295369999999991"/>
    <xsl:with-param name="zoom" select="14"/>
  </xsl:call-template>
</xsl:template>
</xsl:transform>

```

Scala

```

import scala.math._

case class Tile(x: Int, y: Int, z: Short){
  def toLatLon = new LatLonPoint(
    toDegrees(atan(sinh(Pi * (1.0 - 2.0 * y.toDouble / (1<<z))))),
    x.toDouble / (1<<z) * 360.0 - 180.0,
    z)
  def toURI = new java.net.URI("http://tile.openstreetmap.org/"+z+"/"+x+"/"+y+".png")
}

case class LatLonPoint(lat: Double, lon: Double, z: Short){
  def toTile = new Tile(
    ((lon + 180.0) / 360.0 * (1<<z)).toInt,
    ((1 - log(tan(toRadians(lat)) + 1 / cos(toRadians(lat))) / Pi) / 2.0 * (1<<z)).toInt,
    z)
}

//Usage:
val point = LatLonPoint(51.51202, 0.02435, 17)
val tile = point.toTile
// ==> Tile(65544, 43582, 17)
val uri = tile.toURI
// ==> http://tile.openstreetmap.org/17/65544/43582.png

```



```

function osmTileRef iLat, iLong, iZoom --> part path
  local n, xTile, yTile
  put (2 ^ iZoom) into n
  put (iLong + 180) / 360 * n into xTile
  multiply iLat by (pi / 180) -- convert to radians
  put ((1 - ln(tan(iLat) + 1 / cos(iLat)) / pi) / 2) * n into yTile
  return "/" & iZoom & "/" & trunc(xTile) & "/" & trunc(yTile)
end osmTileRef

function osmTileCoords xTile, yTile, iZoom --> coordinates
  local twoPzoom, iLong, iLat, n
  put (2 ^ iZoom) into twoPzoom
  put xTile / twoPzoom * 360 - 180 into iLong
  put pi - 2 * pi * yTile / twoPzoom into n
  put "n1=" && n
  put 180 / pi * atan(0.5 * (exp(n) - exp(-n))) into iLat
  return iLat & comma & iLong
end osmTileCoords

```

Mathematica

```

Deg2Num[lat_, lon_, zoom_] :=
  {IntegerPart[(2^(-3 + zoom))*(180 + lon))/45], IntegerPart[2^(-1 + zoom)*(1 - Log[Sec[Degree*lat] + Tan[Degree*lat]]/Pi)]}

```

```

Num2Deg[xtile_, ytile_, zoom_] :=
  {ArcTan[Sinh[Pi*(1 - 2*(ytile/2^zoom))]]/Degree, (xtile/2^zoom)*360 - 180} // N

```

Tcl

First of all, you need to use the package `map::slippy` from `Tcllib`:

```
package require map::slippy
```

lat/lon to tile number

```
map::slippy geo 2tile [list $zoom $lat $lon]
```

tile number to lat/lon

```
map::slippy tile 2geo [list $zoom $row $col]
```

Pascal

(translated from the Pythoncode above to Pascal)

coordinates to tile numbers

```
uses {...}, Math;
{...}
var
  zoom: Integer;
  lat_rad, lat_deg, lon_deg, n: Real;
begin
  lat_rad := DegToRad(lat_deg);
  n := Power(2, zoom);
  xtile := Trunc(((lon_deg + 180) / 360) * n);
  ytile := Trunc((1 - (ln(Tan(lat_rad) + (1 / Cos(lat_rad)))) / Pi)) / 2 * n);
end;
```

tile numbers to coordinates

```
uses {...}, Math;
{...}
var
  zoom: Integer;
  lat_rad, lat_deg, lon_deg, n: Real;
begin
  lat_rad := DegToRad(lat_deg);
  n := Power(2, zoom);
  xtile := Trunc(((lon_deg + 180) / 360) * n);
  ytile := Trunc((1 - (ln(Tan(lat_rad) + (1 / Cos(lat_rad)))) / Pi)) / 2 * n);
end;
```

R

coordinates to tile numbers

```
deg2num<-function(lat_deg, lon_deg, zoom){
  lat_rad <- lat_deg * pi /180
  n <- 2.0 ^ zoom
  xtile <- floor((lon_deg + 180.0) / 360.0 * n)
  ytile = floor((1.0 - log(tan(lat_rad) + (1 / cos(lat_rad)))) / pi) / 2.0 * n)
  return( c(xtile, ytile))
# return(paste(paste("http://a.tile.openstreetmap.org", zoom, xtile, ytile, sep="/"), ".png", sep=""))
}
```

Subtiles

If you're looking at tile x,y and want to zoom in, the subtiles are (in the next zoom-level's coordinate system):

2x, 2y	2x + 1, 2y
2x, 2y + 1	2x + 1, 2y + 1

Similarly, zoom out by halving x and y (in the previous zoom level)

Tools

- [Online X,Y <-> lat/long conversion](#) [↗](#) ([PHP source](#) [↗](#))
- Same as above plus Tiles preview and direct link to Bigmap [↗](#)
- Javascript Example: Tilesname WebCalc V1.0 [↗](#)
- Geo-OSM-Tiles: a Perl module that calculates tile numbers along with a script that downloads map tiles [↗](#)
- Kachelbrowser [↗](#)
- [File:Lat lon.odt](#) feuille de calcul openoffice (sheet)

References

- http://code.google.com/apis/maps/documentation/overlays.html#Google_Maps_Coordinates [↗](#)
- <http://cfis.savagexi.com/articles/2006/05/03/google-maps-deconstructed> [↗](#)
- "Google Map" projection, see [Spatialreference.org](http://spatialreference.org) [\[1\]](#) [↗](#)
- [OSM mailing list](#) [↗](#) referring to this page.
- [Setting up TMS](#)
- [TMS specification](#) [↗](#) from the [OSGeo Foundation](#) [↗](#)

(note: *Slippy tiles and Google map tiles count tile 0,0 down from the top-left of the tile grid; the TMS spec specifies tiles count up from 0,0 in the lower-left!*)

Categories: [Suggested merges](#) | [Slippy map](#) | [Technical](#) | [Perl](#) | [Java](#) | [C++](#)

This page was last modified on 7 March 2012, at 23:00.

Content is available under [Creative Commons Attribution-ShareAlike 2.0 license](#).

[Privacy policy](#) | [About OpenStreetMap Wiki](#) | [Disclaimers](#)

