Work Instructions

Number: **017**

Name: **Blurring coordinates**

Title: Methods for blurring spatial reference systems on the EU Pollinator Hub

Status: Approved

Version: 1.0

Date: 2023-07-09

Authors: Michael Rubinigg

This document is intended for the regulation of processes within the EU Pollinator Hub. It is published under the CC BY 4.0 licence. You may distribute, transform, adapt and build upon the material in any medium or format, including for commercial purposes, as long as you give credit to the author.

Table of contents

1 Document history 3
1.1 Release 3
1.2 Change history 3
2 Abbreviations 4
3 Introduction 5
4 Validity 6
4.1 Area 6
4.2 Time 6
4.3 Related Documents 6
5 Procedure 7
5.1 Geographical coordinates 7
6 References 8
7 Annexes 9
7.1 Randomisation of geographical coordinates 9
7.2 Reverse geocoding 11

1 Document history

1.1 Release

1.1.1 Drafted

Submitted on 2023-07-10 by Michael Rubinigg, quality manager.

1.1.2 Reviewed

Reviewed on 2023-07-11 by Noa Simòn Delso, general manager.

1.1.3 Approved

Approved on 2023-07-11 by Michael Rubinigg, quality manager.

1.2 Change history

n.a.

2 Abbreviations

n.a.	not available
OSM	OpenStreetMap
WI	Work Instructions

3 Introduction

The document describes the method of randomly blurring data from spatial reference systems.

4 Validity

4.1 Area

The validity of this document is limited to the process of randomly blurring data from spatial reference systems.

4.2 Time

The period of validity of this WI begins on 2023-07-11 and ends with its revocation.

4.3 Related Documents

n.a.

5 Procedure

5.1 Geographical coordinates

5.1.1 Deterministic processing

5.1.1.1 Removal of a decimal place

- For data in the World Geodetic System, version 84 (WGS84, EPSG:4326) round both the latitude and the longitude of a geographical coordinate down to the n^{th} decimal point that corresponds to the desired maximum range by which a position might changes as a result of this operation (refer to Table 1 for rough estimates).

Table 1. Approximate length (m) corresponding to various longitudiunal differences (degrees) at different latitudes on the earth surface, calculated with the formula $\psi = \rho \arccos(\sin \varphi_1 \sin \varphi_1 \cos(\vartheta_1-\vartheta_2) + co s\varphi_1 \cos \varphi_2)$ where ψ is the length difference in m, ρ is the approximate earth's radius in m, φ_1 and φ_2 are the radians of the latitudinal posirtions 1 and 2, ϑ_1 and ϑ_2 are the radians of the longitudinal positions 1 and 2 in degrees. Note that this is a simplified formula that assumes the earth to be perfectly spherical and the radius to be 6373000 m. The approximate length (m) in latitudinal differences corrresponds to the longintudinal differences (degrees) at latitude 0.

	Longitudinal difference in degrees					
latitude	**10**	**1**	**0,1**	**0,01**	**0,001**	**0,0001**
90	0	0	0	0	0	0
80	192911	19315	1931	193	19	2
70	380002	38042	3804	380	38	4
60	555619	55614	5561	556	56	6
50	714439	71497	7150	715	71	7
40	851622	85207	8521	852	85	9
30	962972	96328	9633	963	96	10
20	1045063	104522	10452	1045	105	10
10	1095358	109540	10954	1095	110	11
0	1112298	111230	11123	1112	111	11

5.1.2 Randomisation

5.1.2.1 Transposing a coordinate by a random angle and distance

- Completely randomised blurring (transposing by a random angle and a random distance from the original position) is achieved by using one of the scripts provided in section 7.1.

5.1.3 Replacing coordinates with less specific information

5.1.3.1 Replacing coordinates with OSM objects through reverse geocoding

- Completely randomised blurring (transposing by a random angle and a random distance from the original position) is achieved by using one of the scripts provided in section 7.1.

6 References

There are no sources in the current document.

7 Annexes

7.1 Randomisation of geographical coordinates

7.1.1 php

The following script in the programming language php transforms geographical coordinates with a given lateral (`lat`) and longtundinal position (`lon`) by a randomly generated angle in degrees (`angle_degrees`) and a randomly generated distance in km (`distance_km`) to randomised geographical coordinates with a given lateral (`new_lat`) and longtundinal position (`new_lon`).

```
function transpose_position(lat, lon, angle_degrees, distance_km) {

        lat_rad = deg2rad(lat);
        lon_rad = deg2rad(lon);

        angle_rad = deg2rad(angle_degrees);

        new_lat_rad = asin(sin(lat_rad) * cos(distance_km / 6371) + cos(lat_rad) *
        sin(distance_km / 6371) * cos(angle_rad));

        new_lon_rad = lon_rad + atan2(sin(angle_rad) * sin(distance_km / 6371) * cos(lat_rad),
        cos(distance_km / 6371) - sin(lat_rad) * sin(new_lat_rad));

        new_lat = rad2deg(new_lat_rad);
        new_lon = rad2deg(new_lon_rad);

        return new_lat, new_lon;
}
```

7.1.2 Python

The following script in the programming language Python transforms geographical coordinates with a given lateral (`lat`) and longtundinal position (`lon`) by a randomly generated angle in degrees (`angle_degrees`) and a randomly generated distance in km (`distance_km`) to randomised geographical coordinates with a given lateral (`new_lat`) and longtundinal position (`new_lon`).

```
from geopy.distance import geodesic
import math
import random

def transpose_position(lat, lon, angle_degrees, distance_km):

        lat_rad = math.radians(lat)
        lon_rad = math.radians(lon)

        angle_rad = math.radians(angle_degrees)

        new_lat_rad = math.asin(math.sin(lat_rad) * math.cos(distance_km / 6371) +
        math.cos(lat_rad) * math.sin(distance_km / 6371) * math.cos(angle_rad))

        new_lon_rad = lon_rad + math.atan2(math.sin(angle_rad) * math.sin(distance_km / 6371) *
        math.cos(lat_rad), math.cos(distance_km / 6371) - math.sin(lat_rad) *
        math.sin(new_lat_rad))

        new_lat = math.degrees(new_lat_rad)
        new_lon = math.degrees(new_lon_rad)

        return new_lat, new_lon
```

7.1.3 MariaDB

The following procedure for MariaDB updates geographical coordinates with a given lateral (`posttransform.latitude`) and longtundinal position (`posttransform.longitude`) by a randomly created angle in degrees (`@angle_degrees`) and a randomly created distance in km (`@distance_km`), which depends on the maximum and minimum values (`distance_from`, `distance_to`) passed to the procedure to randomised geographical coordinates with a given lateral (`@new_lat`) and longtundinal position (`@new_lon`) in the columns `posttransform.latitude_rand` and `posttransform.longitude_rand`. Note that table and coilun names nee to be adapted to the actual names.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `postransform`(IN `distance_from` DECIMAL(6,3), IN
`distance_to` DECIMAL(6,3))
BEGIN

DECLARE done INT DEFAULT FALSE;
DECLARE record ROW(id BIGINT, latitude DECIMAL(12,9), longitude DECIMAL(12,9));

DECLARE cur CURSOR FOR SELECT id, latitude, longitude FROM postransform ORDER BY id;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=TRUE;

OPEN cur;

READ_LOOP: LOOP

       FETCH cur into record;

    SET @angle_degrees = FLOOR(0 + RAND() * (360));
    SET @distance_km = FLOOR(distance_from + RAND() * (distance_to - distance_from +1));

    SET @lat_rad = RADIANS(record.latitude);
    SET @lon_rad = RADIANS(record.longitude);
    SET @angle_rad = RADIANS(@angle_degrees);

    SET  @new_lat_rad  =  ASIN(SIN(@lat_rad)  *  COS(@distance_km  /  6371)  +  COS(@lat_rad)  *
SIN(@distance_km / 6371) * COS(@angle_rad));
    SET  @new_lon_rad  =  @lon_rad  +  ATAN2(SIN(@angle_rad)  *  SIN(@distance_km  /  6371)  *
COS(@lat_rad), COS(@distance_km / 6371) - SIN(@lat_rad) * SIN(@new_lat_rad));

    SET @new_lat = DEGREES(@new_lat_rad);
    SET @new_lon = DEGREES(@new_lon_rad);

    UPDATE postransform SET latitude_rand=@new_lat;
    UPDATE postransform SET longitude_rand=@new_lon;

    IF done THEN
       LEAVE READ_LOOP;
    END IF;

END LOOP;

CLOSE cur;

END$$
DELIMITER ;
```

7.2 Reverse geocoding

7.2.1 OpenStreetMap (OSM)

The following Python script updates geographical information on a position in a MariaDB table with information obtained by accessing the Nominatim reverse geocoding API in compliance with the Nominatim usage policy. The script reads the record identifier and the geographical positions contained in the columns with the name id, latitude and longitude, respectively, from a csv export of a MariaDB table and updates a MariaDB table with information in geocodejson format obtained from the request based on the record identifier. The information is written to the columns name (name of the OSM object, usually the municipality), state (higher level administrative unit of the object), country (country in which the object is located), country_code (2 letter country code), latitude_obj (latitude provided for the OSM object), longitude_obj (latitude provided for the OSM object). The path of the source file, the host name of your account on the database server, the user name of your account on the database server, the password of the account on the database server, the name of the database on the database server, the e-mail address if required for compliance with Nominatim usage policy, the alpha-2 language code of your preferred language and the name of the table to which the information should be written have to be entered by the user through dialog windows.

```
# Title: NominatimReverse.py
# Author: Rubinigg Michael
# Contributor: Rubinigg Michael
# Abstract: reads latitudes and longitudes from a csv export of a MariaDB query and replaces a
MariaDb table with results obtained from the reverse geocoding request to the Nominatim RestAPI
(https://nominatim.org/release-docs/develop/api/Overview/)
# Version 1.0
# Release date: 2023-07-10
# Place: Graz, Austria, EU
# Company: dr Mag. rer. nat. Michael Rubinigg
# Programming Language: Python 3.11
# Rights:  Creative Commons Attribution 4.0 (CC-BY)

import csv
import time
import mysql.connector
import requests
from tkinter import filedialog, simpledialog
import tkinter

def get_string_input(title, prompt):
    # Create the Tkinter window
    window = tkinter.Tk()
    window.withdraw()

    # Prompt the user for string input
    user_input = simpledialog.askstring(title, prompt)

    # Close the Tkinter window
    window.destroy()

    # Return the user input
    return user_input

def get_source_file():
    window = tkinter.Tk()
    window.withdraw()
```

```
    print("Select the input csv file.")
    input_file_path = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])

    # Close the Tkinter window
    window.destroy()

    if not input_file_path:
        print("No input file selected. Exiting.")
        return

    return input_file_path

# Function get_api_data makes a request to Nominatim API and get the address information
def get_api_data(latitude, longitude, email, lan):
    # Access the RestAPI (refer to https://nominatim.org/release-docs/develop/api/Overview/ for
further information)
    api_endpoint = f'https://nominatim.openstreetmap.org/reverse'
    print()
    params = {
        'lat': latitude,
        'lon': longitude,
        'zoom': 10,
        'format': 'geocodejson',
        'addressdetails': 1,
        'accept-language': lan+';q=0.9,en;q=0.8',
        'email': email
    }

    response = requests.get(api_endpoint, params=params)

    if response.status_code == 200:
        return response.json()
    else:
        print(f"Failed to fetch data from API. Status code: {response.status_code}")
        return None


# Function to extract and write data to CSV
def extract_and_write(db, table, row_identifier, json_data):
    if json_data:

        geocoding_json       =       json_data.get('features',       [])[0].get('properties',
{}).get('geocoding', {})
        geometry_json = json_data.get('features', [])[0].get('geometry', {}).get('coordinates',
{})

        query = f'UPDATE {table} set '

        geocoding_key = ['name', 'state', 'country', 'country_code']

        for key in geocoding_key:
            value = geocoding_json.get(key, 'NULL')

            if value is None:
                value = 'NULL'

            query += f'{key}="{value}", '

        if geometry_json:
            latitude = geometry_json[1]
            longitude = geometry_json[0]
        else:
            latitude = 'NULL'
            longitude = 'NULL'

        id = row_identifier

        query += f' latitude_obj={latitude}, longitude_obj={longitude} WHERE id={id}'

        print(query)

        try:
            # Create a cursor object to execute SQL queries
            cursor = db.cursor()

            # Execute the query
            cursor.execute(query)
```

```
            db.commit()

        except mysql.connector.Error as err:
            print(f"Error: {err}")

        finally:
            # Close the cursor
            if 'cursor' in locals():
                cursor.close()


# Main script
def main():
    input_file_path = get_source_file()

    try:

        host = get_string_input('Host', 'Enter the host name of your account on the database
server:')
        user = get_string_input('User', 'Enter the user name of your account on the database
server:')
        password = get_string_input('Password', 'Enter the password of the account on the
database server:')
        database = get_string_input('Database', 'Enter the name of the database on the database
server:')
        email = get_string_input('Email', 'Enter your e-mail address if required for compliance
with Nominatim usage policy:')
        lan = get_string_input('Language', 'Enter the alpha-2 language code of your preferred
language:')
        table = get_string_input('Table', 'Enter the name of the table to which the information
should be written:')

        # Establish a connection to the MariaDB server
        db = mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            database=database
        )

        if db.is_connected():
            print("Connected to MariaDB!")

        # Open input and output CSV files
        with open(input_file_path, 'r') as input_csv:
            csv_reader = csv.DictReader(input_csv)

            for row in csv_reader:

                # Check if latitudine and longitudine are present and not NULL
                lat = row['latitude']
                lon = row['longitude']

                if lat != 'NULL' and lon != 'NULL':

                    row_identifier = row['id']
                    location_info = get_api_data(lat, lon, email, lan)

                    if location_info.get('error') != 'Unable to geocode':
                        extract_and_write(db, table, row_identifier, location_info)

                        time.sleep(1)

    except mysql.connector.Error as err:
        print(f"Error: {err}")

    finally:
        # Close the cursor and connection when done
        if 'connection' in locals() and db.is_connected():
            db.close()
            print("Connection closed.")


if __name__ == "__main__":
    main()
```