PRODUCT SPECIFICATIONS

Thank you for buying a GeoPostcodes product, we appreciate your business. Hereafter you will find some technical information on GeoPostcodes database and how to import the data in your application.

USE OF GEOPOSTCODES DATABASE IS SUBJECT TO OUR TERMS AND CONDITIONS. YOU MAY NOT USE THE DATABASE IN ANY MANNER UNLESS YOU ACCEPT THE TERMS AND CONDITIONS OF THE LICENSE. IF YOU DO NOT ACCEPT ALL OF THE TERMS AND CONDITIONS OF THE LICENSE, DO NOT USE THE DATABASE IN ANY MANNER.

GEOPOSTCODES DATASETS

Our standard datasets contain all localities, administrative regions and ZIP/postal codes* of a country, including business and administrative codes, neighborhoods, suburbs and streets when available. Additional information such as time zones, elevation and administrative standard codes (ISO, FIPS, HASC, NUTS, National Statistics) are also provided.

All data are georeferenced using the WGS84 datum and can be easily imported into any software, database or GIS system.

This is the perfect product if you need to update your contact list, validate forms, build a store locator, display locations on a map, create statistics reports and much more.

STREET+ DATASETS

Available for some countries, STREET+ is a comprehensive and geocoded list of street names for each city. This is the perfect tool to build a geocoder, check and improve the quality of your addresses or enhance your GIS system.

PACKAGES

If you need more data, we have bulk packages. This is a great opportunity to get a complete region at a discounted price.

ADMINISTRATIVE DIVISIONS DATABASE

In addition to our datasets of standard places, our *GeoPostcodes Administrative Divisions Database* "GADB" is the most comprehensive and up-to-date source for administrative regions. This database contains all official divisions for each country in the world with statistical reference codes.

FILE FORMATS

Our data are provided in several text and GIS formats which can easily be imported into any software or database management system. All available formats are stored in a single zipped file.

CSV

CSV (Comma-separated values) data is one of the most ubiquitous file formats, most spreadsheets and databases can easily read native CSV files. Its simple format can be edited in a text editor. Many programming languages have special libraries to read and write CSV files, therefore making it a great medium for exchanging large quantities of data.

SHAPEFILE

ESRI Shapefile format (.shp) is a popular geospatial vector data format which can be imported into most geographic information system software (GIS) like ArcGIS, Grass, Quantum GIS, PostGIS and many others.

KML

The Keyhole Markup Language (.kml) is another popular geographic format using XML notation and maintained by Google. It can be used with Google products like Google Earth or Google Maps, as well as with many other software programs.

ASC

We provide an alternative normalized ASCII version in CSV format where all accents and diacritics have been removed. This version can be used to facilitate searching or sorting.

EXTENSION	ENCODING	DESCRIPTION
.CSV	UTF-8	Comma separated value
.ASC	ASC	Comma separated value unformatted ASCII
.KML	XML UTF-8	Google Keyhole Markup Language
.SHP	Binary	Shapefile geographic data
.DBF	dBase IV	Shapefile attributes table
.SHX	Binary	Shapefile index file
.PRJ	Text	Shapefile projection format

TABLES AND FIELDS DESCRIPTION

The data in our files are provided in a consistent structure, however, depending on the country, some fields may remain void.

LOCALITIES AND STREETS (GEOPC_XX_PLACES.CSV / GEOPC_XX_STREETS.CSV)

FIELD NAME	DATA TYPE	DESCRIPTION
ISO	Char(2)	ISO 3166-1 Country code
Country	Char(50)	Country name
Language	Char(2)	ISO 639-1 language code
ID	Integer	Record identifier
Region1	Char(80)	Administrative region level 1
Region2	Char(80)	Administrative region level 2
Region3	Char(80)	Administrative region level 3
Region4	Char(80)	Administrative region level 4
Locality	Char(80)	Locality name
Postcode	Char(15)	ZIP/Postal code
Suburb	Char(80)	Locality subdivision
Street	Char(100)	Street name
Range	Char(50)	Street numbers range
Latitude	Double	Latitude coordinates in WGS84
Longitude	Double	Longitude coordinates in WGS84
Elevation	Integer	Elevation in meters
ISO2	Char(10)	ISO 3166-2 Region code
FIPS	Char(10)	NGA Geopolitical code (formerly FIPS PUB 10-4)
NUTS	Char(12)	European subdivision code
HASC	Char(12)	Hierarchical administrative subdivision code
STAT	Char(20)	National statistics/census code
Timezone	Char(30)	Time zone name (Olson)
UTC	Char(10)	Coordinated Universal Time
DST	Char(10)	Daylight Saving Time

ADMINISTRATIVE REGIONS GADB (GEOPC_XX_REGIONS.CSV)

FIELD NAME	DATA TYPE	DESCRIPTION
ISO	Char(2)	ISO 3166-1 Country code
Country	Char(50)	Country name
Language	Char(2)	ISO 639-1 language code
Level	Integer	Administrative level
Type	Char(50)	Type of administrative unit
Name	Char(80)	Administrative division name
Region1	Char(80)	Administrative region level 1
Region2	Char(80)	Administrative region level 2
Region3	Char(80)	Administrative region level 3
Region4	Char(80)	Administrative region level 4
ISO2	Char(10)	ISO 3166-2 Region code
FIPS	Char(10)	NGA Geopolitical code (formerly FIPS PUB 10-4)
NUTS	Char(12)	European subdivision code
HASC	Char(12)	Hierarchical administrative subdivision code
STAT	Char(20)	National statistics/census code

BUSINESS & ADMINISTRATIVE (GEOPC_XX_BUSINESS.CSV)

FIELD NAME	DATA TYPE	DESCRIPTION
ISO	Char(2)	ISO 3166-1 Country code
Country	Char(50)	Country name
Language	Char(2)	ISO 639-1 language code
ID	Integer	Record identifier
Region1	Char(80)	Administrative region level 1
Region2	Char(80)	Administrative region level 2
Region3	Char(80)	Administrative region level 3
Region4	Char(80)	Administrative region level 4
Entity	Char(100)	Entity name
Postcode	Char(15)	ZIP/Postal code

DATA IMPORT

Below you will find how to import GeoPostcodes datasets into the most common DBMS.

MYSQL

Use the following SQL code to create the table structure and indexes in MySQL:

```
DROP TABLE IF EXISTS GeoPC_Places;
DROP TABLE IF EXISTS GeoPC_Business;
DROP TABLE IF EXISTS GeoPC_Regions;

CREATE TABLE GeoPC_Places (
  ISO varchar(2) NOT NULL,
  Country varchar(50) NOT NULL,
  Language varchar(2) NOT NULL,
  ID bigint(20) NOT NULL,
  Region1 varchar(80),
  Region2 varchar(80),
  Region3 varchar(80),
  Region4 varchar(80),
  Postcode varchar(15),
  Locality varchar(80),
  Suburb varchar(80),
  Street varchar(100),
  `Range` varchar(50),
  Latitude double,
  Longitude double,
  Elevation integer,
  ISO2 varchar(10),
  FIPS varchar(10),
  NUTS varchar(12),
  HASC varchar(12),
  STAT varchar(20),
  Timezone varchar(30),
  UTC varchar(10),
  DST varchar(10),
  PRIMARY KEY (Language, ID)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE GeoPC_Business (
  ISO varchar(2) NOT NULL,
  Country varchar(50) NOT NULL,
  Language varchar(2) NOT NULL,
  ID bigint(20) NOT NULL,
  Region1 varchar(80),
  Region2 varchar(80),
  Region3 varchar(80),
  Region4 varchar(80),
```

```
  Postcode varchar(15),
  Entity varchar(100),
  PRIMARY KEY (Language, ID)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE GeoPC_Regions (
  ISO varchar(2) NOT NULL,
  Language varchar(2) NOT NULL,
  Level smallint(8) NOT NULL,
  Type varchar(50) NOT NULL,
  Name varchar(80) NOT NULL,
  Country varchar(50) NOT NULL,
  Region1 varchar(80),
  Region2 varchar(80),
  Region3 varchar(80),
  Region4 varchar(80),
  ISO2 varchar(10),
  FIPS varchar(10),
  NUTS varchar(12),
  HASC varchar(12),
  STAT varchar(20)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Use the following SQL code to import the CSV data in MySQL:

```
LOAD DATA INFILE '/path/GeoPC_xx_Places.csv' INTO TABLE GeoPC_Places
FIELDS ERMINATED BY ';' ESCAPED BY '\\' IGNORE 1 LINES;

LOAD DATA INFILE '/path/GeoPC_xx_Business.csv' INTO TABLE
GeoPC_Business FIELDS TERMINATED BY ';' ESCAPED BY '\\' IGNORE 1 LINES;

LOAD DATA INFILE '/path/GeoPC_xx_Regions.csv' INTO TABLE
GeoPC_Regions FIELDS ERMINATED BY ';' ESCAPED BY '\\' IGNORE 1 LINES;
```

POSTGRES

Use the following SQL code to create the table structure and indexes in Postgres:

```
DROP TABLE IF EXISTS geopc_places;
DROP TABLE IF EXISTS geopc_business;
DROP TABLE IF EXISTS geopc_regions;

CREATE TABLE geopc_places
(
  iso character varying(2) NOT NULL,
  country character varying(50) NOT NULL,
  language character varying(2) NOT NULL,
  id integer NOT NULL,
  region1 character varying(80),
  region2 character varying(80),
  region3 character varying(80),
  region4 character varying(80),
  postcode character varying(15),
  locality character varying(80),
  suburb character varying(80),
  street character varying(100),
  range character varying(50),
  latitude double precision,
  longitude double precision,
  elevation integer,
  iso2 character varying(10),
  fips character varying(10),
  nuts character varying(12),
  hasc character varying(12),
  stat character varying(20),
  timezone character varying(30),
  utc character varying(10),
  dst character varying(10),
  CONSTRAINT geopc_places_pkey PRIMARY KEY (language, id)
)
WITH (
  OIDS=FALSE
);

CREATE TABLE geopc_business
(
  iso character varying(2) NOT NULL,
  country character varying(50) NOT NULL,
  language character varying(2) NOT NULL,
  id integer NOT NULL,
  region1 character varying(80),
  region2 character varying(80),
  region3 character varying(80),
  region4 character varying(80),
  postcode character varying(15),
```

```
  entity character varying(100),
  CONSTRAINT geopc_business_pkey PRIMARY KEY (language, id)
)
WITH (
  OIDS=FALSE
);

CREATE TABLE geopc_regions
(
  iso character varying(2) NOT NULL,
  language character varying(2) NOT NULL,
  level smallint NOT NULL,
  type character varying(50) NOT NULL,
  name character varying(80) NOT NULL,
  country character varying(50) NOT NULL,
  region1 character varying(80),
  region2 character varying(80),
  region3 character varying(80),
  region4 character varying(80),
  iso2 character varying(10),
  fips character varying(10),
  nuts character varying(12),
  hasc character varying(12),
  stat character varying(20)
)
WITH (
  OIDS=FALSE
);
```

Use the following SQL code to import the CSV data in Postgres:

```
COPY geopc_places FROM '/path/GeoPC_xx_Places.csv' WITH DELIMITER AS ';'
CSV HEADER ENCODING 'UTF-8';

COPY geopc_business FROM '/path/GeoPC_xx_Business.csv' WITH DELIMITER AS
';' CSV HEADER ENCODING 'UTF-8';

COPY geopc_regions FROM '/path/GeoPC_xx_Regions.csv' WITH DELIMITER AS ';'
CSV HEADER ENCODING 'UTF-8';
```

SQL SERVER

Use the following SQL code to create the table structure and indexes in SQL Server:

```
DROP TABLE GeoPC_Places, GeoPC_Business, GeoPC_Regions;
GO

CREATE TABLE GeoPC_Places (
  ISO VARCHAR(2) NOT NULL,
  Country VARCHAR(50) NOT NULL,
  Language VARCHAR(2) NOT NULL,
  ID BIGINT NOT NULL,
  Region1 VARCHAR(80),
  Region2 VARCHAR(80),
  Region3 VARCHAR(80),
  Region4 VARCHAR(80),
  Postcode VARCHAR(15),
  Locality VARCHAR(80),
  Suburb VARCHAR(80),
  Street VARCHAR(100),
  `Range` VARCHAR(50),
  Latitude DECIMAL(10,6),
  Longitude DECIMAL(10,6),
  Elevation INT,
  ISO2 VARCHAR(10),
  FIPS VARCHAR(10),
  NUTS VARCHAR(12),
  HASC VARCHAR(12),
  STAT VARCHAR(20),
  Timezone VARCHAR(30),
  UTC VARCHAR(10),
  DST VARCHAR(10),
  CONSTRAINT GeoPC_Places_pkey PRIMARY KEY CLUSTERED (Language ASC,
ID ASC) ON [PRIMARY]
);
GO

CREATE TABLE GeoPC_Business (
  ISO VARCHAR(2) NOT NULL,
  Country VARCHAR(50) NOT NULL,
  Language VARCHAR(2) NOT NULL,
  ID BIGINT NOT NULL,
  Region1 VARCHAR(80),
  Region2 VARCHAR(80),
  Region3 VARCHAR(80),
  Region4 VARCHAR(80),
  Postcode VARCHAR(15),
  Entity VARCHAR(100),
  CONSTRAINT GeoPC_Business_pkey PRIMARY KEY CLUSTERED (Language ASC,
ID ASC) ON [PRIMARY]
);
```

```
GO

CREATE TABLE GeoPC_Regions (
  ISO VARCHAR(2) NOT NULL,
  Language VARCHAR(2) NOT NULL,
  Level SMALLINT NOT NULL,
  Type VARCHAR(50) NOT NULL,
  Name VARCHAR(80) NOT NULL,
  Country VARCHAR(50) NOT NULL,
  Region1 VARCHAR(80),
  Region2 VARCHAR(80),
  Region3 VARCHAR(80),
  Region4 VARCHAR(80),
  ISO2 VARCHAR(10),
  FIPS VARCHAR(10),
  NUTS VARCHAR(12),
  HASC VARCHAR(12),
  STAT VARCHAR(20)
);
GO
```

Use the following SQL code to import the CSV data in SQL Server. Please note that CSV file must first be converted in UTF-16 encoding format as BULK INSERT command does not support UTF-8.

```
BULK INSERT GeoPC_Places FROM 'Path\GeoPC_xx_Places.csv' WITH (FIRSTROW =
2, FIELDTERMINATOR = ';', ROWTERMINATOR = '\n', DATAFILETYPE
= 'widechar') GO

BULK INSERT GeoPC_Business FROM 'Path\GeoPC_xx_Business.csv' WITH (FIRSTROW
= 2, FIELDTERMINATOR = ';', ROWTERMINATOR = '\n', DATAFILETYPE
= 'widechar') GO

BULK INSERT GeoPC_Regions FROM 'Path\GeoPC_xx_Regions.csv' WITH (FIRSTROW =
2, FIELDTERMINATOR = ';', ROWTERMINATOR = '\n', DATAFILETYPE
= 'widechar') GO
```

ORACLE

Use the following SQL code to create the table structure and import the CSV data in **Oracle**:

```
DROP TABLE IF EXISTS GeoPC_Places;
DROP TABLE IF EXISTS GeoPC_Business;
DROP TABLE IF EXISTS GeoPC_Regions;

CREATE TABLE GeoPC_Places (
  ISO VARCHAR2(2) NOT NULL,
  Country VARCHAR2(50) NOT NULL,
  Language VARCHAR2(2) NOT NULL,
  ID INTEGER NOT NULL,
  Region1 VARCHAR2(80),
  Region2 VARCHAR2(80),
  Region3 VARCHAR2(80),
  Region4 VARCHAR2(80),
  Postcode VARCHAR2(15),
  Locality VARCHAR2(80),
  Suburb VARCHAR2(80),
  Street VARCHAR2(100),
  Range VARCHAR2(50),
  Latitude NUMBER(10,6),
  Longitude NUMBER(10,6),
  Elevation INTEGER,
  ISO2 VARCHAR2(10),
  FIPS VARCHAR2(10),
  NUTS VARCHAR2(12),
  HASC VARCHAR2(12),
  STAT VARCHAR2(20),
  Timezone VARCHAR2(30),
  UTC VARCHAR2(10),
  DST VARCHAR2(10),
  CONSTRAINT GeoPC_Places_pkey PRIMARY KEY (Language, ID)
)
ORGANIZATION EXTERNAL
( DEFAULT DIRECTORY xtern_data_dir
  ACCESS PARAMETERS
  ( RECORDS DELIMITED BY NEWLINE
    FIELDS TERMINATED BY ';'
  )
  LOCATION ('/path/GeoPC_xx_Places.csv')
);

CREATE TABLE GeoPC_Business (
  ISO VARCHAR2(2) NOT NULL,
  Country VARCHAR2(50) NOT NULL,
  Language VARCHAR2(2) NOT NULL,
  ID INTEGER NOT NULL,
  Region1 VARCHAR2(80),
  Region2 VARCHAR2(80),
```

```
  Region3 VARCHAR2(80),
  Region4 VARCHAR2(80),
  Postcode VARCHAR2(15),
  Entity VARCHAR2(100),
  CONSTRAINT GeoPC_Business_pkey PRIMARY KEY (Language, ID)
)
ORGANIZATION EXTERNAL
( DEFAULT DIRECTORY xtern_data_dir
  ACCESS PARAMETERS
  ( RECORDS DELIMITED BY NEWLINE
    FIELDS TERMINATED BY ';'
  )
  LOCATION ('/path/GeoPC_xx_Business.csv')
);

CREATE TABLE GeoPC_Regions (
  ISO VARCHAR2(2) NOT NULL,
  Language VARCHAR2(2) NOT NULL,
  Level INTEGER NOT NULL,
  Type VARCHAR2(50),
  Name VARCHAR2(80),
  Country VARCHAR2(50),
  Region1 VARCHAR2(80),
  Region2 VARCHAR2(80),
  Region3 VARCHAR2(80),
  Region4 VARCHAR2(80),
  ISO2 VARCHAR2(10),
  FIPS VARCHAR2(10),
  NUTS VARCHAR2(12),
  HASC VARCHAR2(12),
  STAT VARCHAR2(20)
)
ORGANIZATION EXTERNAL
( DEFAULT DIRECTORY xtern_data_dir
  ACCESS PARAMETERS
  ( RECORDS DELIMITED BY NEWLINE
    FIELDS TERMINATED BY ';'
  )
  LOCATION ('/path/GeoPC_xx_Regions.csv')
);
```

SQLITE

Use the following SQL code to create the table structure and indexes in SQLite:

```
DROP TABLE IF EXISTS GeoPC_Places;
DROP TABLE IF EXISTS GeoPC_Business;
DROP TABLE IF EXISTS GeoPC_Regions;

CREATE TABLE GeoPC_Places (
  ISO CHAR(2) NOT NULL,
  Country CHAR(50) NOT NULL,
  Language CHAR(2) NOT NULL,
  ID INTEGER NOT NULL,
  Region1 CHAR(80),
  Region2 CHAR(80),
  Region3 CHAR(80),
  Region4 CHAR(80),
  Postcode CHAR(15),
  Locality CHAR(80),
  Suburb CHAR(80),
  Street CHAR(100),
  Range CHAR(50),
  Latitude NUMERIC,
  Longitude NUMERIC,
  Elevation INTEGER,
  ISO2 CHAR(10),
  FIPS CHAR(10),
  NUTS CHAR(12),
  HASC CHAR(12),
  STAT CHAR(20),
  Timezone CHAR(30),
  UTC CHAR(10),
  DST CHAR(10),
  PRIMARY KEY (Language, ID)
);

CREATE TABLE GeoPC_Business (
  ISO CHAR(2) NOT NULL,
  Country CHAR(50) NOT NULL,
  Language CHAR(2) NOT NULL,
  ID INTEGER NOT NULL,
  Region1 CHAR(80),
  Region2 CHAR(80),
  Region3 CHAR(80),
  Region4 CHAR(80),
  Postcode CHAR(15),
  Entity CHAR(100),
  PRIMARY KEY (Language, ID)
);

CREATE TABLE GeoPC_Regions (
```

```
  ISO CHAR(2) NOT NULL,
  Language CHAR(2) NOT NULL,
  Level INTEGER NOT NULL,
  Type CHAR(50),
  Name CHAR(80),
  Country CHAR(50),
  Region1 CHAR(80),
  Region2 CHAR(80),
  Region3 CHAR(80),
  Region4 CHAR(80),
  ISO2 CHAR(10),
  FIPS CHAR(10),
  NUTS CHAR(12),
  HASC CHAR(12),
  STAT CHAR(20)
);
```

Use the following SQL code to import the CSV data in SQLite:

```
LOAD DATA INLINE '/path/GeoPC_xx_Places.csv' INTO TABLE GeoPC_Places FIELDS
TERMINATED BY ';' LINES TERMINATED BY '\n' IGNORE 1 LINES;

LOAD DATA INLINE '/path/GeoPC_xx_Business.csv' INTO TABLE
GeoPC_Business FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n' IGNORE
1 LINES;

LOAD DATA INLINE '/path/GeoPC_xx_Regions.csv' INTO TABLE
GeoPC_Regions FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n' IGNORE
1 LINES;
```

MS-ACCESS

Follow the steps below to import the CSV data in Access:

```
Open or create a new Access database
Open menu File/Get External Data/Import
Select "Text Files (*.txt;*.csv;*.tab;*.asc)" as type of file
Browse directories and select your file (GeoPC_*.csv)
Select Delimited, click on Advanced and select Unicode (UTF-8) as Code Page
Click Next, select Semicolon as delimiter and {none} as Text qualifier
Tick First row contains field names and click on Finish
```

MS-EXCEL

Follow the steps below to import the CSV data in Excel:

```
In menu File/Open, select "Text Files (*.prn;*.txt;*.csv)" as type of file
Browse directories and select your file (GeoPC_*.csv)
In import wizard, select Delimited type and 65001: Unicode (UTF-8) encoding
Click Next, select Semicolon as delimiter and {none} as Text qualifier
```

Please note that the following limits applies to Excel sheets:

- Excel 2003 and earlier : 65536 rows
- Excel 2007/2010/2013 : 1048576 rows