COVER PAGE

DELIVERABLE

Project Acronym: i-locate

Grant Agreement number: 621040

Project Title: Indoor/outdoor LOCation and Asset management Through open gEodata

D2.3 – Geospatial database with pilot-relevant data and open data connectors

(accompanying report)

Revision: Final

Authors:

Martino Salvetti (TRILOGIS)

Stefano Piffer (TRILOGIS)

Lucian Brancovean (INDSOFT)

Catalin Popa (INDSOFT)

Ramona Candea (INDSOFT)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	**Public**	**X**
C	**Confidential, only for members of the consortium and the Commission Services**	

REVISION HISTORY AND STATEMENT OF ORIGINALITY

Revision History

Revision	Date	Author	Organisation	Description
V0.9	09/12	Lucian	INDSOFT	Structure of the document
V1.0	12/12	Lucian	INDSOFT	Architecture, Connectors, Resulting data
V1.1	15/12	Giuseppe	TRILOGIS	Full review
V1.2	15/12	Lucian	INDSOFT	Minor correction
Final	22/12/	Irene	TRILOGIS	Quality Check

Statement of originality:

1 List of references

Number	Full reference
1	i-locate deliverable D.2.2 – Analysis of open data repositories. Available online from: http://www.i-locate.eu/public-deliverables/
2	GTFS format https://developers.google.com/transit/gtfs/
3	Ogr2ogr tool http://www.gdal.org/ogr2ogr.html
4	GDAL License http://opensource.org/licenses/mit-license.php
5	
6	

2 Table of Acroyms

Acronym	Descriptiobn
CSV	Comma-Separated Values
XML	Extensible Markup Language
SHP	ESRI Shapefile
GML	Geography Markup Language
KML	Keyhole Markup Language
KMZ	Zipped KML files with a .kmz extension
XLS	Microsoft Excel file format
ODS	Operational Data Store
TSV	Tab-Separated Values
JSON	JavaScript Object Notation
GeoJSON	Open standard format for encoding collections of simple geographical features using JSON
PDF	Portable Document Format
RDF	Resource Description Framework
GDAL	Geospatial Data Abstraction Library
OGC	Open Geospatial Consortium
WMS	Web Mapping Service
WFS	Web Feature Service

Table of content

3 Introduction

In order to allow the i-locate toolkit to access data from various open data repositories, a set of connectors have been developed. In practice, these are modules of a set of tools that allow importing geospatial data from open data repositories into a PostgreSQL / PostGIS database. More specifically, each tool has been implemented in Java and it can called from a method call or it be invoked as a command-line program.

The full set of details regarding the open data repositories that these connectors read from has been provided within deliverable D.2.2 – Analysis of open data repositories [1].

Since the aforementioned tools have been officially released as deliverable D.2.3 of type "Other" it has been decided to edit this document, as accompanying report, which provides a short description of the tools developed.

3.1 Requirements

For the tools to run it requires the following software installed:

- PostgreSQL database, version 9.3
- PostGIS extension for PostgreSQL, version 2.1
- Java Runtime Environment, version 7
- Ogr2ogr tool from GDAL 1.11 (Geospatial Data Abstraction Library)

It should be noted that any operating system that supports this software can be used.

4 Architecture

As mentioned before, the connectors are developed as Java software and command-line applications (it is possible to invoke each component from the command line). Once invoked, each software component performs the required data transfer, it writes a log and then exits. The possibility to run it from the command line, it allows the creation of scripts that can automatically schedule each component at intervals, in order to keep the database in sync with the various data sources. The data repository(ies) each component has to access, are described within a configuration file, as well as the database each component has to write to.

Internally, each connector is made up of several connectors which use some generic services required, for instance, to provide access to the database, to download files from a web server, etc. Each connector is implemented by a Java class, as are the services, whose detailed description is provided below.

4.1 Project plan

There are two specific services provided by each connector, which can be reached trough the `ServiceProvider` class.

The first, called **`DatabaseService`**, allows access to two features, namely: `queryDatabase` and `insertIntoDatabase`. This approach ensures that connectors have a high degree of independence from the database configuration and connection logic.

The second service, called **`Ogr2ogrService`**, provides an abstract view to access the third party tool ogr2ogr [3] through an `ogr2ogrImportIntoPostgreSQL` operation. The latter calls the ogr2ogr tool to directly import files into Postgres without the need to explicitly parse the data, connect to the database, perform insert operation, and so on. As detailed later in this document, this allow supporting a range of different formats, in a simple manner.

It should be noted that the services are eventually configured separately from the connectors. This allows the connectors to share –if needed- the same configuration information. A typical example is the fact that, through this approach, they can all share the same user and password to connect to the database.

4.2 Connectors

It is worth noting that the open data repositories used in i-locate are managed by third parties and therefore expose data in several different formats. For this reason the following different connectors have been developed for different data formats, and used where needed.

- The **`ConnectorSHP`** class is used to import data presented in the ESRI Shapefile (SHP) format. Since parsing this format is relatively complicated, a simpler solution was found through use of third party tools already doing the parsing job. As a result, this connector downloads the shapefiles and then calls the aforementioned `ogr2ogr` tool to import them in the database.
- The **`ConnectorGML`** class is used to import data saved in GML format (Geography Markup

Language) by the Open Geospatial Consortium. Similarly to the shapefile format, GML has a fairly complex syntax therefore `ogr2ogr` is used to import this data in the database.

- The **`ConnectorCSV`** class is used to import data stored as CSV (Comma-Separated Value) format. It downloads the file from the source repository, it parses it, and inserts it in the database through the `DatabaseService`. The class has some configuration options to identify the column with the geographic coordinates and the field separator.
- The **`ConnectorGTFS`** class is used to import data presented in the GTFS (General Transit Feed Specifications) format. The class downloads the GTFS zip file from the server, it extracts the files from the archive, and it then uses the CSV connector to parse the stops.txt file and to insert the data in the database.
- The **`ConnectorKMZ`** class is used to import data saved in the KMZ format (Zipped Keyhole Markup Language). The software downloads the KMZ file and unpacks it. It then uses the `ogr2ogr` tool to import the KML file in the database.
- The **`ConnectorJSON`** class is used to import data presented in JSON format. It downloads the JSON (JavaScript Object Notation) file, parses it, and then it inserts data in the database using the `DatabaseService`.

This modular architecture allows development of new connectors in the future, for different data formats, with minimal impact to the existing connectors, and it allows reusing significant amounts of code.

4.3 Configuration

All components share the same configuration file name, i.e. `connectors.xml`. The file contains the address of the data repository that has to be imported to PostgreSQL. If the application is run from command line, then the configuration file has to be saved in the same directory as the jar application file.

The application's configuration file is an XML document which contains the following nodes. The root node is named `<repositories>`. All other nodes are contained within this node. This nodes hold a number of `<entry>` nodes, one for each file that will be imported. Additionally, it contains a node of type `<destinationDatabase>` which stores details of the database, we well as a node of type `<ogr2ogr>` with the parameters be used within the `ogr2ogr2` tool.

The default configuration file `connectors.xml` can be used as an example for further data repositories since the file provided by default it is preconfigured to include the open data repositories identified in the document D2.2.

The following sections provide the full set of details regarding the configuration parameters.

4.3.1 Database

- The tag `<destinationDatabase></destinationDatabase>` specifies the driver name, the URL of the PostgreSQL database together with username and password needed to access it:

- The tag `<driverName>org.postgresql.Driver</driverName>` specifies the driver name used to connect to the PostgreSQL database.
- The tag `<URL>jdbc:postgresql://localhost:5432/geocoords</URL>` specifies the details of the database whose parameters are:
  - The host, (which should replace „localhost" in the example).
  - The port on which the database runs (which should replace „5432" in the example).
  - The name of the database into which the data will be inserted (which should replace „geocoords" in the example).
- The tag `<username>postgres</username>` specifies the username used to connect to PostgreSQL.
- The tag `<password>postgres</password>` specifies the password used togheter with the username.
- The tag `<logEveryNRecords>100</logEveryNRecords>` specifies the number of inserted records for which the application writes a log in the console.

4.3.2 Geospatial Data Abstraction Library – ogr2ogr

The ogr2ogr tool can be publicly downloaded from http://www.gdal.org/index.html. However, it should be noted that the tool does not support SHP files by default. For this reason, it has been decided to rely – within i-locate- on the ogr2ogr version contained in QGIS, which supports that file SHP format. This version of the tool can be downloaded from http://www.qgis.org/.

The `<ogr2ogr>` tag has the following parameters:

- The tag `<path>c:\Program Files\QGIS Chugiak\bin\ogr2ogr.exe</path>` specifies the path where the tool is installed.
- The tag `<databaseHost>localhost</databaseHost>` specifies the host where the PostgreSQL database is run.
- The tag `<databasePort>5432</databasePort>` specifies the port on which the database host runs.
- The tag `<databaseName>geocoords</databaseName>` specifies the name of the database into which the data will be inserted.
- The tag `<databaseUsername>postgres</databaseUsername>` specifies the username used to connect to PostgreSQL.

- The tag `<databasePassword>postgres</databasePassword>` specifies the password used together with the username.

4.3.3 Connectors Configuration

4.3.3.1 Connector SHP

To use this connector, an `<entry></entry>` node has to be defined in the configuration file which contains the following structure:

- The tag `<name>` contains the name of the log file that will contain information about the import process.
- `The tag <URI>` specifies the address from where the file will be downloaded.
- The tag `<connector>` contains the fully qualified package name. For this connector, this tag contains the following text: `eu.ilocate.portal.connectors.ConnectorSHP`.
- The tag `<ogr2ogrImportMethod>` can have two values:
  - `overwrite`: the imported data will overwrite the data already present within the database.
  - `append`: the imported data will be added to the database, keeping the data previously stored in the database.

4.3.3.2 ConnectorGML

To use this connector, an `<entry></entry>` node has to be defined in the configuration file which contains the following structure:

- The tag `<name>` contains the name of the log file that will contain the information about the import process.
- The tag `<URI>` specifies the address from where the file will be downloaded.
- The tag `<connector>` contains the fully qualified package name. For this connector, this tag contains the following text: `eu.ilocate.portal.connectors.ConnectorGml`
- The tag `<ogr2ogrImportMethod>` can have two values:
  - `overwrite`: the imported data will overwrite the data already present within the database.
  - `append`: the imported data will be added to the database, keeping the data previously stored in the database.

4.3.3.3 ConnectorCSV

To use this connector, an `<entry></entry>` node has to be defined in the configuration file which contains the following structure:

- The tag `<URI>` contains the name of the log file that will contain information about the import process.
- The tag `<connector>` contains the fully qualified package name. For this connector, this tag contains the following text: `eu.ilocate.portal.connectors.ConnectorCsv`.
- The tag `<separator>` specifies the delimiter from the CSV file: comma (,) or semicolon (;)
- The tag `<longitudeLatitudeOneColumn>` can have two values: true or false and it is related to the following two tags. If it is not present, it will signal that the source file does not have geographical data as coordinates and will not look for it. If it is not present the `<longitudeColumn>`, `<latitudeColumn>` and `<longitudeLatitudeColumns>` parameters will not be checked, they are not necessary.
- The tag `<longitudeColumn>` has to be used only if `<longitudeLatitudeOneColumn>` exists and is set to false and it specifies the index of the column (starting from 1) that contains the longitude coordinate.
- The tag `<latitudeColumn>` has to be used only if `<longitudeLatitudeOneColumn>` exists and is set to false and it specifies the index of the column (starting from 1) that contains the latitude coordinate.
- The tag `<longitudeLatitudeColumns>` has to be used only if `<longitudeLatitudeOneColumn>` exists and is set to true and it specifies the index of the column (starting from 1) that contains both the latitude and longitude coordinate.
- The tag `<replaceQuotAmpersand>` can take two possible values: true or false. If true, then the file contains quotes" as &quote that needs to be replaced with „" and ampersand as &amp that needs to be replaced with &.
- The tag `<tableName>` specifies the name of the table that will be created for the data to be inserted. The application transforms this table name to lowercase consequently creating a table in PosgtgreSQL with a lowercase name.

4.3.3.4 ConnectorGTFS

To use this connector, an `<entry></entry>` node has to be defined in the configuration file which contains the following structure:

- The tag `<URI>` contains the name of the log file that will contain information about the import process.
- The tag `<connector>` contains the fully qualified package name. For this connector, this tag contains the following text: `eu.ilocate.portal.connectors.ConnectorGtfs`
- The tag `<separator>` specifies the delimiter from the CSV file: comma (,) or semicolon (;)
- The tag `<longitudeLatitudeOneColumn>` This takes two values: true or false.
- The tag `<longitudeColumn>` It counts only if `<longitudeLatitudeOneColumn>` is set to false. It specifies the index of the column (starting from 1) that contains the longitude coordinate.
- The tag `<latitudeColumn>` It counts only if `<longitudeLatitudeOneColumn>` is set to false. It specifies the index of the column (starting from 1) that contains the latitude coordinate.
- The tag `<longitudeLatitudeColumns>` It counts only if `<longitudeLatitudeOneColumn>` is set to true. It specifies the index of the column (starting from 1) that contains both the latitude and longitude coordinate.
- The tag `<replaceQuotAmpersand>` can have two values, true or false. If true, then the file contains quotes as &quote that needs to be replaced with „" and ampersand as &amp that needs to be replaced with &.
- The tag `<tableName>` is name of the table that will be created and where the data will be be inserted into. The application transforms this table name to lowercase consequently creating a table in PosgtgreSQL with lowercase file name.

4.3.3.5 ConnectorKMZ

To use this connector, an `<entry></entry>` node has to be defined in the configuration file which contains the following structure:

- The tag `<name>` contains the name of the log file that will contain information about the import process.
- The tag `<URI>` specifies the address from where the file will be downloaded.
- The tag `<connector>` contains the fully qualified package name. For this connector, this tag contains the following text: `eu.ilocate.portal.connectors.ConnectorKmz`
- The tag `<ogr2ogrImportMethod>` can have take values:

- `overwrite`: the imported data will overwrite the data already present within the database.
- `append`: the imported data will be added to the database, keeping the data previously stored in the database.

4.3.3.6 ConnectorJSON

To use this connector, an `<entry></entry>` node has to be defined in the configuration file which contains the following structure:

- The tag `<URI>` specifies the name of the log file that will contain information about the import process.
- The tag `<connector>` contains the fully qualified package name. For this connector, this tag contains the following text: `eu.ilocate.portal.connectors.ConnectorJson`
- `The tag <tableName>` contains the name of the table that will be created for the data to be inserted into. The application transforms this table name to lowercase consequently creating a table in PosgtgreSQL with lowercase name.
- The tag `<longitudeNodeName>` specifies the name of the node that contains the longitude coordinate from JSON file.
- The tag `<latitudeNodeName>` specifies the name of the node that contains the latitude coordinate from JSON file.
- The tag `<jsonPath>` specifies the path from the JSON file with the array that contains the values that will be inserted into database.

4.4 Execution

The program inserts the data downloaded from the data sources identified within the configuration file into the PostgreSQL database. In order to run the program, the following command have to be run through a command inside the console:

```
java -jar [-Dlog4j.configuration="file:log4j.xml"] connectors-0.0.1-SNAPSHOT-jar-with-dependencies.jar
```

The system provides by default an extensive logging feature based on use of log4j configuration. However, this can be overridden with a special parameter passed as `log4j.configuration="file:log4j.xml"` switch (optional). The configuration switch can be used to pass the path of the `log4j.xml` file which is used as configuration file for the log4j API. In this case, the path is the same as the jar file. It is worth noting that the name file of the –jar file that contains the application is `connectors-0.0.1-SNAPSHOT-jar-with-dependencies.jar`.

5 Resulting database

The use of the connector tools requires the creation of several tables within the PostgreSQL database. Typically, each entry in the configuration file results in a separate table, as follows:

- The table `dhmosia_wifi` is generated for the repository MH-OPN-02: Greece – Public Wi-Fi Hot-Spots. The data contains a description of the Wi-Fi access point, their location expressed in latitude and longitude. The table also contains information regarding the municipality to which the access points belong to and the IP addresses of the Wi-Fi points.
- The table `dhmosia_kthria` is generated for the repository MH-OPN-03: Greece – Public Buildings. The data contains the information on the buildings, including like the region they are located, the county they are located, their size, type, address, number, the municipality they belong to and the location of each building through their latitude and longitude.
- The table `verkeersregelinstallatie_locaties` is generated for the repository UMC-OPN-02: Utrecht – Building DataBase. The file contains geographic points that draws the road. The roads that include also lane separation.
- The table `strade` is generated for the repository RO-OPN-02: Rovereto – roads. The data contains streets codes, geographic coordinates, street type (whether it is a road, square, alley, avenue, footpath etc.) together with the name of the streets.
- The table `rovereto` is generated for the repository RO-OPN-03: Rovereto – Impaired people facilities. The data contains the name of the service, street address, zip code, municipality, province, country to which it belongs to, the Internet site, altitude, some accessibility codes and some of their properties (description of the service), opening hours, last modified date and geographic coordinates.
- The table `stops` is generated for the repository RO-OPN-04: Rovereto – Transportation facilities. The data provided here is conforming to the GTFS (General Transit Feed Specifications) standard, and it comes in a ZIP package. Each file contains transit information including the data provider (the agency), timetable, stops and routes. The stop files contains geographical information.
- The table `line_features` is generated for the repository VE-OPN-02: Province of Rome – Road Map. The data contains street codes, names, categories.
- The table `ultimorilevamento` is generated for the repository VE-OPN-03: Province of Rome – Viability. The data contains name of the street, coordinates of the streets, date and time of the collection, the flow of the vehicles. This appears to be filled in real-time when an event happens, and is empty most of the time.
- The table `accesspointexita` is generated for the repository VE-OPN-04: Wifi map of province of Rome. The data contains the name of the access points, their geographic

coordinates, street address, the municipality where they belong to and the type (public or private).

- The tables `hotellistmalta, restaurants, heritagesitesjan14, maltabays` are generated for the repository SJH-OPN-02: Malta – Facilities. The available datasets are:
    - Hotels in Malta: the data contains information about the hotels: geographic coordinates, hotel ranking, their name, their address, the towns they are situated in, their fax and telephone number, e-mail and website.
    - Restaurants in Malta: the data contains information about the restaurants: their name, address, geographic coordinates, telephone number, website, spoken language and opening hours.
    - Heritage Sites: the data contains information about the heritage sites: description, type (museum, garden, temples, church etc.), period, location, geographic coordinates, opening days, the number of the bus that can be took, contact, telephone number, e-mail address, spoken language.
    - Maltese Bays: the data contains information about bays: name, geographic coordinates and spoken language.
- The table `italypublichealthagencies` is generated for the repository ITA-OPN-01: Italy – Public Health Agencies. The data contains the year, the ID of the region and its name (E.g. Piedmont), the code of the health agency, the name of the health agency, and its full address.
- The table `italypharmaceuticalvendingmachine` is generated for the repository ITA-OPN-02: Italy – Automatic pharmaceutical vending machines. The data contains the ID of the vending machine, the name of the place of the company where the machine is located and its full address, the VAT number of the company, the city (including name of the city and unique ID), validity date range, type of vending machine, role of the company holding the vending machine (e.g. leaser), position in latitude and longitude.
- The table `italyparapharmacies` is generated for the repository ITA-OPN-03: Italy – Public para-pharmacies. The data contains the unique ID, the name of the shop, the full address, its VAT number, other information regarding the code of the city where it is located, starting date when the was licensed and eventually the position in latitude and longitude.
- The table `italypublicpharmacies` is generated for the repository ITA-OPN-04: Italy – Public pharmacies. The data contains the unique ID, the name of the shop, the corresponding health agency ID, the full address, its VAT number, other information regarding the code of the city where it is located, starting date when the was licensed and eventually the position in latitude and longitude.