DB Energie GPS comparator - Documentation

Mikoláš Fromm

02.08.2023

1 Introduction

This document describes the DB Energie GPS comparator application. The application is used to compare the GPS data from the DB Energie invoicing system with the data from the GPS system of the vehicle. The whole application is written in C# and is equipped with WinForms interface for easy use.

Motivation

My motivation for this topic is my part-time job in a company which operates locomotives in Germany (on Deutsche Bahn infrastructure). Deutsche Bahn (or DB from now on) regularly send energy invoices that full of mistakes where they claim the locomotive was spending energy in Germany, even though in reality the locomotive was outside Germany completely, for example in Hungary. Therefore is then needed to compare all timestamps from the DB invoices with the location data from the locomotive GPS system. This is a very time-consuming process, therefore I decided to automate it.

Goals

Major goal was to create an application capable of checking the correctness of the data from the invoice and separating the correct from the incorrect. Minor goals were also to automatically generate a refund request file and to automatically reassign all energy consumptions to all operators that were using the locomotive at the given time.

2 Application subdivision

The project is separated into the following sections:

- Wrappers
    - DB Energie wrapper
    - GPS wrapper
    - LokoUsage wrapper
- Data structures
- Data processing
    - Comparator
    - Exporter
- WinForm interface

2.1 Wrappers

Wrappers are used to communicate with or parse data from external systems. The main reason for this separation is to make the application more modular and easier to maintain. At the moment, none of the sources is offering any kind of API, therefore all the wrappers require uploading a .csv or .xslx file manually, but in case it changes, only the single wrapper will need to be modified and the rest of the application will remain the same.

All the wrappers define their own Interface, with which the core of the application communicates. This interface is then implemented by the wrapper itself.

2.1.1 DB Energie wrapper

```
public interface IDBE_wrapper
{
    void GetAllEntriesFromDBE();

    List<DbeEntry> Entries { get; }

    HashSet<LocoId> LocosIncluded { get; }

    public Dictionary<int, LocoId> LocoIdForGivenColumn { get; }
}
```

This wrapper currently has two implementations available:

1. ```
   // reading attachment for DBE invoice
   public class DBE_wrapper : IDBE_wrapper
   ```

2. ```
   // reading CSV export from BahnStromPortal
   public class DBE_abstimmung_wrapper : IDBE_wrapper
   ```

where the first one parses the input .csv file, which is attached to the DB Energie invoice and which has one 15 minutes long energy consumption timespan for each locomotive on separate row. It is also to notice that the invoice contains only the locomotives that were spending energy in Germany and that the entries in the attachement are sorted by the datetime, which is later used in evaluation.

The second implementation is used to parse the .csv file exported from the BahnStromPortal, which contains all the energy consumption timespans for all locomotives, even those that were not spending energy in Germany. The entries are also sorted by datetime, but each row contains entry for each locomotive. This information must be saved for later evaluation, therefore the $Dictionary < int, LocoId >$ is used, where the key is the column number and the value is the LocoId of the locomotive in that column.

The application is also prepared for a very big amount of data, therefore the wrapper is not saving all entries to the memory, but only saves the datetime span, locoId and the coordinates of the whole entry. This implies that the data file is read multiple times during the evaluation, but the user memory will be saved.

2.1.2 GPS wrapper

```
public interface IGPS_wrapper
{
    // indexed by each loco, containing sorted dates "from - to" in germany
    Dictionary<LocoId, List<DateSpan>> GetAllTimesInGermany(Dictionary<LocoId,
                                                    GpsLocoFilePath> gpsMapping);
    HashSet<LocoId> LocomotivesWithOutGPS { get; }
}
```

This wrapper is used to parse the GPS data from the locomotive GPS system. It uses output from PosiTrex GPS system, which is able to export all border-crossings which determines exactly when each locomotive has entered / left a specific country. PosiTrex is capable of exporting only a file per locomotive, therefore the implementation requires a filePath to the border-crossing file of each locomotive involved. Then in only reads the file line by line, where one line contains dateFrom, dateTo and the country of the activity, and filters out only the lines containing "Germany" as the country. The result is then saved in the $Dictionary < LocoId, List < DateSpan >>$, where the key is the LocoId and the value is the list of all the timespans when the locomotive was in Germany.

It might also happen that the locomotive has faulty or no GPS system installed and therefore the GPS data is not available. In this case, the locomotive is considered to be in Germany for the whole time and the entry is saved in the $HashSet < LocoId > LocomotivesWithOutGPS$ to later notify the user about this fact.

2.1.3 LokoUsage wrapper

```
public interface ILokoUsage_wrapper
{
    // indexed by locomotive, containing "from - to" and customer name for each time span.
    Dictionary<LocoId, List<CustomerDateSpan>> GetAllCustomers(IEnumerable<LocoId> locomotives);

    IList<string> CustomerNames { get; }
}
```

2.2 Data structures

2.3 Data processing

2.3.1 Comparator

2.3.2 Exporter

```
public interface IExporter
{
    void ExportAndFillTemplate(EvaluationResults evaluationResults,
                               Dictionary<LocoId, List<CustomerDateSpan>> customerDateTimes,
                               IEnumerable<string> customerNames,
                               double price);

    void ExportAndFillTemplate(EvaluationResults evaluationResults);

    void ExportAndFillTemplate(EvaluationResults evaluation,
                               Dictionary<int, LocoId> LocoIdForGivenColumn);

    IExporter AddOutputDir(string outputDir);
}
```

2.4 WinForm interface