



European
Commission

Using binary data serialization for data storage and sharing

Peter Mooney (IE)



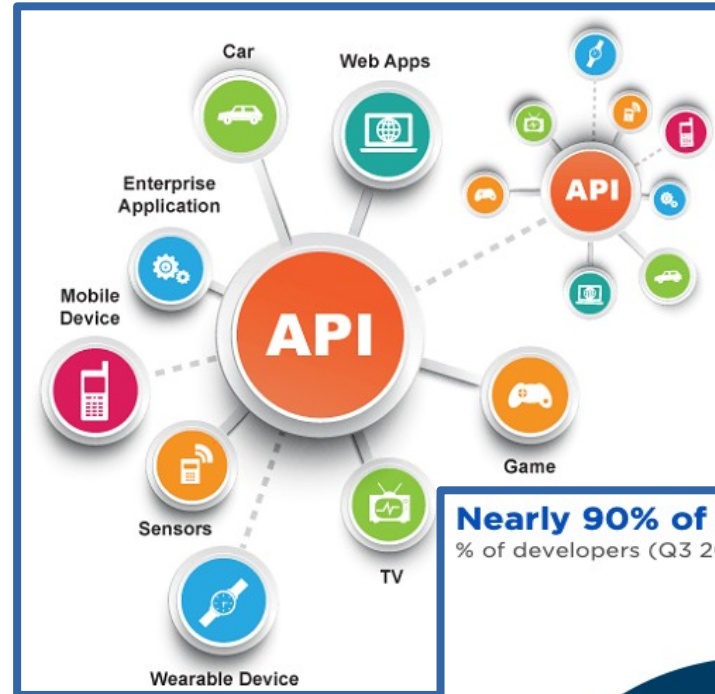
European Commission
Joint Research Center
JRC/B/06

EXPERT CONTRACT
CONTRACT NUMBER - CT-EX2014D166355-104

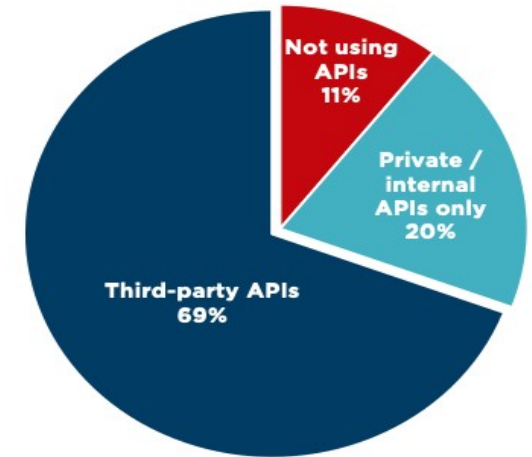
Internet-based (Geo)data storage and sharing requirements are growing every day



Cloud Storage



Nearly 90% of developers use APIs
% of developers (Q3 2020 n=15,299)



Source: SlashData Developer Economics survey 19th edition / DATA



“The best time to plant a tree is yesterday, the next best time is today”

“For the development of many products and services, **data needs to be widely and easily available, easily accessible, and simple to use and process.** Data has become a key factor of production, and the value it creates has to be shared back with the entire society participating in providing the data. This is why **we need to build a genuine European single market for data** - a European data space based on European rules and values.”

“In order to open up key public sector reference data sets for innovation, it shall start the procedure for the adoption of an Implementing act on high-value data sets (Q1 2021) under the Open Data Directive, making these data sets available across the EU for free, in **machine-readable format and through standardised Application Programming Interfaces (APIs)**”

European strategy for data COM/2020/66



Common European data spaces

Rich pool of data
(varying degree of
accessibility)

Free flow of data
across sectors and
countries

Full respect of GDPR

Horizontal
framework for data
governance and data
access



- Technical tools for data pooling and sharing
- Standards & interoperability (technical, semantic)
- Sectoral Data Governance (contracts, licenses, access rights, usage rights)
- IT capacity, including cloud storage, processing and services

Generally, most APIs available today expose services providing XML, JSON, GeoJSON, CSV, etc.

• Advantages

- **Almost universal client tool support** (programming languages, GIS, mobile, etc)
- **Interoperable** (and open data formats)
- **Human readable**
- **Works very effectively for small data sizes**
- **JSON = 'de-facto' standard**

• Disadvantages

- **Poor performance** on larger data sizes
- **Typing:** Does not always impose strict 'typing'
- **Scales poorly** over time and space (data sizes)
- **Not necessarily suitable for cloud infrastructures**

Typical, popular, use-case scenario for API (geodata)

- **Server**

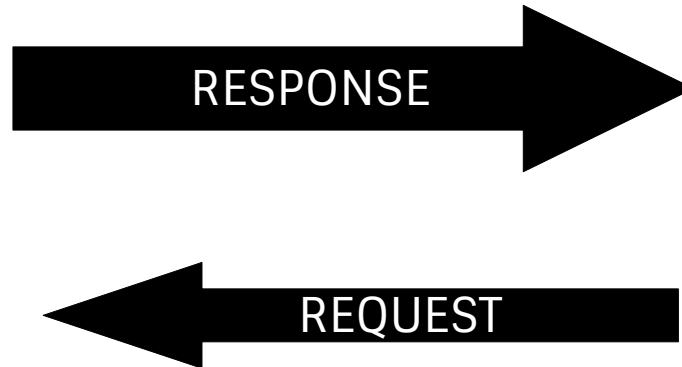
- Receives request
- Prepares response
- Send response (JSON, XML, csv, SHP, etc)

- **Client tools**

- Load response
- Extract or Transform
- Process: Visualise, analyse, integrate, etc.



(Geo)data SERVER – with API



CLIENT(S) – with software

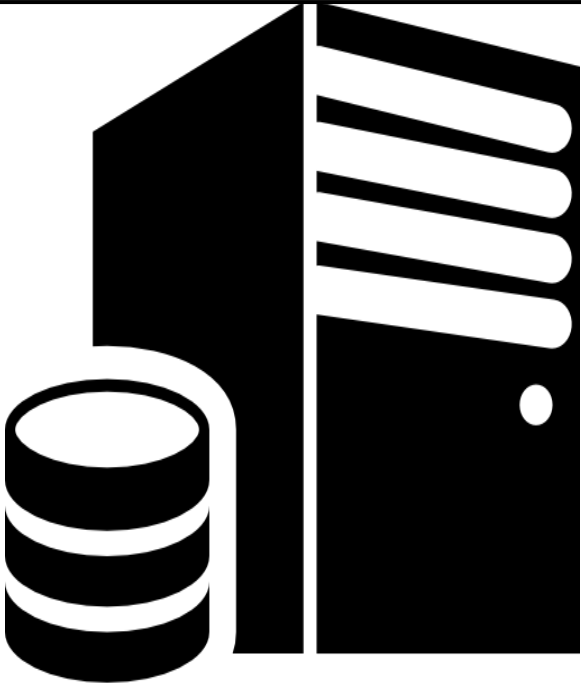
As responses scale the use of popular data formats can introduce many obstacles

- **Server**

- Preparation of complex responses (query times)
- High traffic, network bandwidth (uncontrollable)

- **Client tools**

- Delayed response
- Loading large responses
- Long times: Extracting, transforming, processing large responses

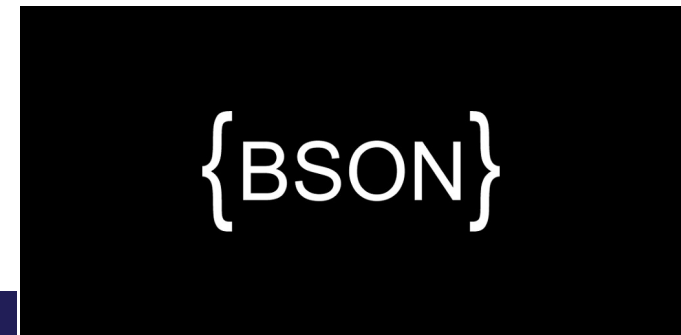
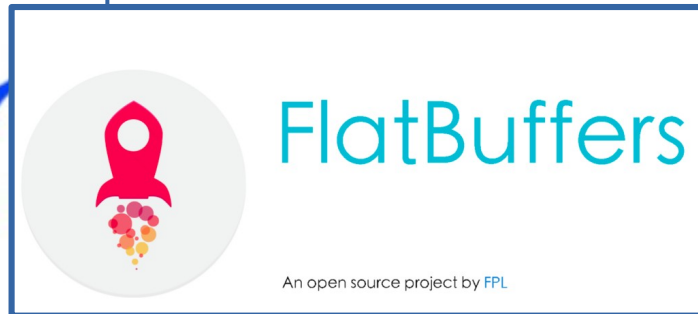
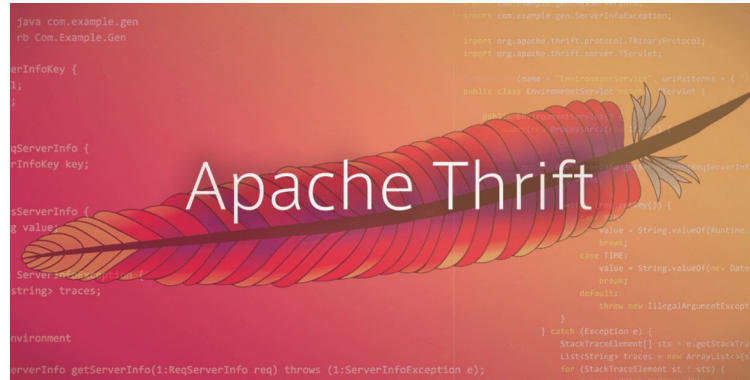


(Geo)data SERVER – with API



CLIENT(S) – with software

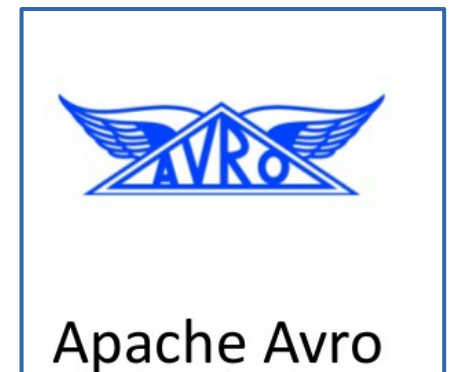
Binary data serialization – LOTS of options to choose from



Experimental setup and focus



- Not just “**time vrs space**” analysis of binary vrs JSON/XML serialization approaches
- Focus on interoperability, usability, scalability
- Open source, open approaches
- Investigate conditions where binary serialization could replace or compliment the ‘de-factor’ standards (JSON, XML, and so on...)
- Google Protocol Buffers, Apache Avro



Experimental setup – Data (1)

- **Experiment 1**

- **A “large” static GIS dataset** – Dr. Alessandro Sarretta’s project
- POINT Geometry
- NLS Finland + OpenStreetMap addresses
- 1.9M features
- GPKG file

- **Experiment 2**

- **An API (OGC Sensor Things API)** – Dr. Simon Jirka’s project
- POINT Geometry
- Aeroplane tracking
- 20,000 features
- JSON response

Experimental setup – Data (2)

- **Experiment 1**

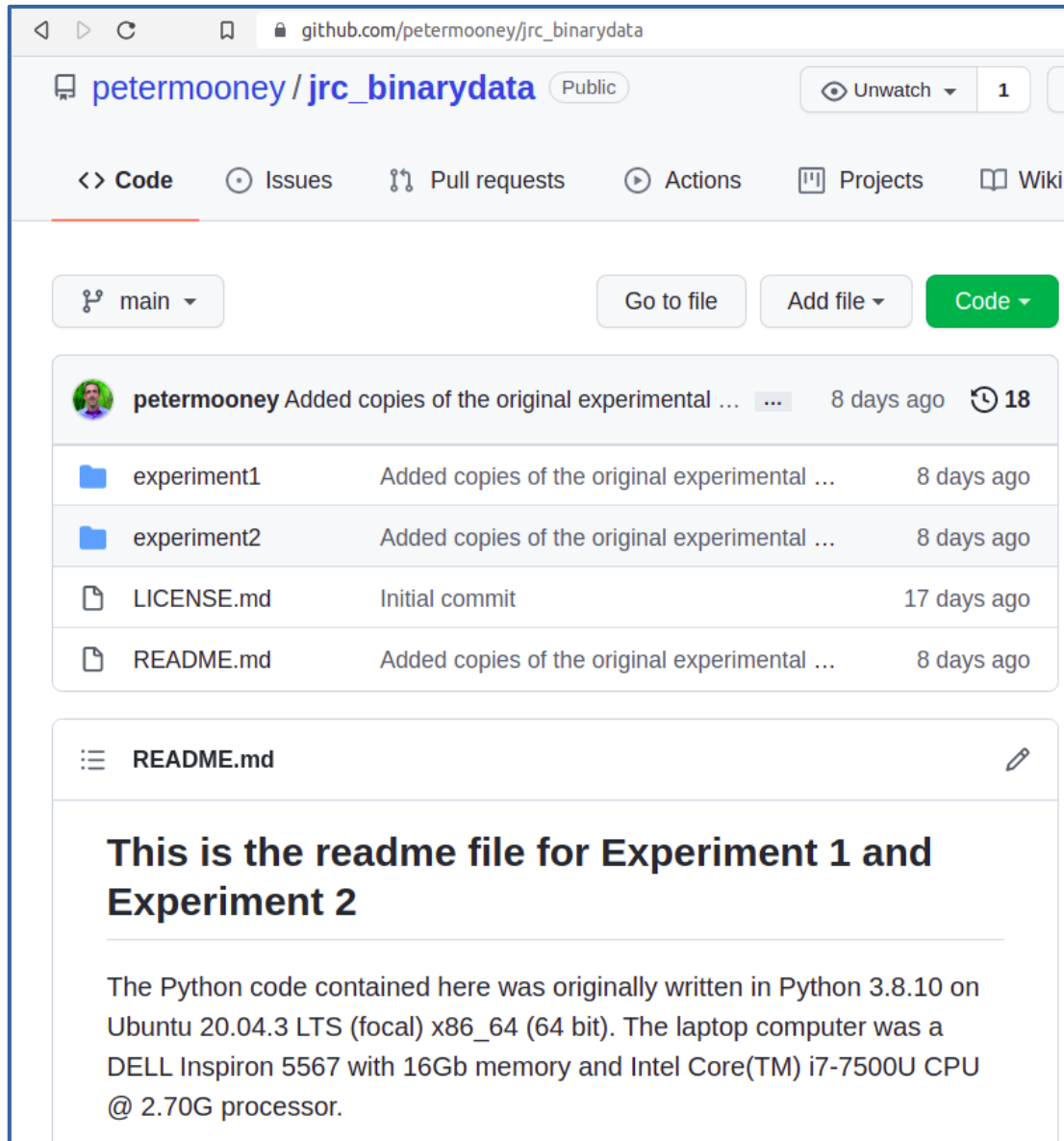
- **A “large” static GIS dataset** – Dr. Alessandro Sarretta’s project
- POINT Geometry
- NLS Finland + OpenStreetMap addresses
- 1.9M features
- GPKG file

- **Experiment 1a**

- **A static GIS dataset** – Dr. Alessandro Sarretta’s project
- POINT Geometry
- **Randomly generated data** (same attribute names and types as original)
- 20,000 features
- GPKG file
- **For reproducibility purposes**

Experimental Setup (Software)

https://github.com/petermooney/jrc_binarydata



github.com/petermooney/jrc_binarydata

petermooney / jrc_binarydata Public

Code Issues Pull requests Actions Projects Wiki

main Go to file Add file Code

petermooney Added copies of the original experimental ... 8 days ago 18

experiment1	Added copies of the original experimental ...	8 days ago
experiment2	Added copies of the original experimental ...	8 days ago
LICENSE.md	Initial commit	17 days ago
README.md	Added copies of the original experimental ...	8 days ago

README.md

This is the readme file for Experiment 1 and Experiment 2

The Python code contained here was originally written in Python 3.8.10 on Ubuntu 20.04.3 LTS (focal) x86_64 (64 bit). The laptop computer was a DELL Inspiron 5567 with 16Gb memory and Intel Core(TM) i7-7500U CPU @ 2.70G processor.



Software setup

- Fully reproducible code (GitHub)
- No “hacks”
- Use open source and widely supported Python libraries only
- Interoperable

Experiment 1 – Binary schemas



```
1 {
2   "name": "FinlandAddresses",
3   "namespace": "FI_Addresses",
4   "type": "record",
5   "fields": [
6     {"name": "addrHouseNumber", "type": ["string","null"]},
7     {"name": "addrStreet", "type": ["string","null"]},
8     {"name": "addrCity", "type": ["string","null"]},
9     {"name": "fid", "type": "int"},
10    {"name": "source", "type": ["string","null"]},
11    {"name": "addrUnit", "type": ["string","null"]},
12    {"name": "fullAddress", "type": ["string","null"]},
13    {"name": "geometry", "type": ["string","null"]}
14  ]
15 }
```

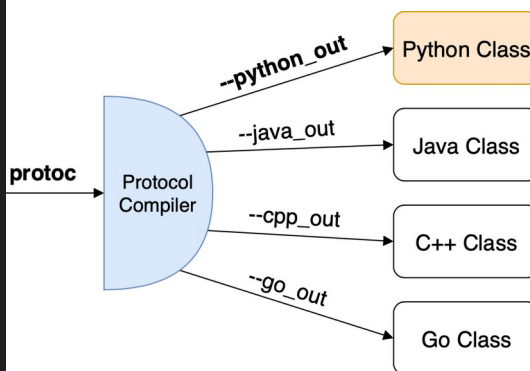
Original GPKG

Layer Properties — original-dataset-experiment1 test-geopackage — Fields

Id	Name	Alias	Type	Type name	Length	Precision	Comment
123 0	fid		qlonglong	Integer64	0	0	
abc 1	addr:houseNumber		QString	String	0	0	
abc 2	addr:street		QString	String	0	0	
abc 3	addr:country		QString	String	0	0	
abc 4	addr:city		QString	String	0	0	
abc 5	source		QString	String	0	0	
abc 6	fullAddress		QString	String	0	0	
abc 7	addr:unit		QString	String	0	0	



```
1 syntax = "proto2";
2 package AddressFI;
3
4 message Address_prop {
5   required string addrHouseNumber = 1;
6   required string addrStreet = 2;
7   required string addrCity = 3;
8   required int32 fid = 4;
9   required string source = 5;
10  required string addrUnit = 6;
11  required string fullAddress = 7;
12  required string geometry = 8;
13 }
14
15 message Address {
16   repeated Address_prop address = 1;
17 }
```



```
1 # -*- coding: utf-8 -*-
2 # Generated by the protocol buffer compiler. DO NOT EDIT!
3 # source: address.proto
4 """Generated protocol buffer code."""
5 from google.protobuf import descriptor as _descriptor
6 from google.protobuf import message as _message
7 from google.protobuf import reflection as _reflection
8 from google.protobuf import symbol_database as _symbol_database
9 # @@protoc_insertion_point(imports)
10
11 _sym_db = _symbol_database.Default()
12
13
14
15
16 DESCRIPTOR = _descriptor.FileDescriptor(
17   name='address.proto',
18   package='AddressFI',
19   syntax='proto2',
```

Experiment 2 – Binary schemas

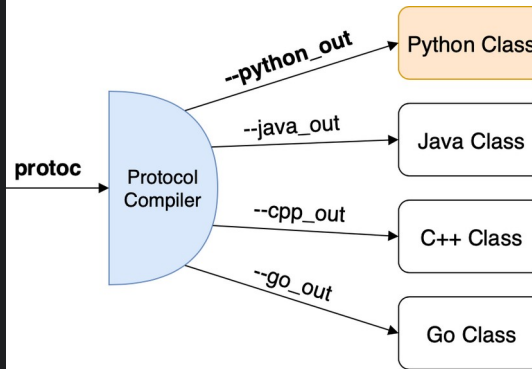
Original API response

```
1 {
2   "name": "Experiment2",
3   "namespace": "Experiment2STA",
4   "type": "record",
5   "fields": [
6
7     {"name": "name", "type": "string"},
8     {"name": "iotid", "type": "string"},
9     {"name": "description", "type": "string"},
10    {"name": "iotsselfLink", "type": "string"},
11    {"name": "things_iot_navigationLink", "type": "string"},
12    {"name": "historical_locations_iot_navigationLink", "type": "string"},
13    {"name": "longitude", "type": "float"},
14    {"name": "latitude", "type": "float"}
15  ]
16 }
17 }
```



```
1 {
2   "value": [{
3     "@iot.id": "0000b162-4d9b-4727-b148-706f5d25219a",
4     "@iot.selfLink": "https://jrc.dev.52north.org/v1.1/Locations(0000b162-4d9b-4727-
5     "name": "Location of 400663",
6     "description": "Somewhere in the sky",
7     "encodingType": "application/vnd.geo+json",
8     "location": {
9       "type": "Point",
10      "coordinates": [7.01165, 51.66806],
11      "crs": {
12        "type": "name",
13        "properties": {
14          "name": "EPSG:4326"
15        }
16      }
17    },
18    "Things@iot.navigationLink": "https://jrc.dev.52north.org/v1.1/Locations(0000b16
19    "HistoricalLocations@iot.navigationLink": "https://jrc.dev.52north.org/v1.1/Loca
20  ], {
```

```
1 syntax = "proto2";
2
3 package Experiment2Data;
4
5 message Experiment2_prop {
6   required string name = 1;
7   required float latitude = 2;
8   required float longitude = 3;
9   required string iotid = 4;
10  required string iotsselfLink = 5;
11  required string description = 6;
12  required string historicalLink = 7;
13  required string thingsLink = 8;
14 }
15 }
16 message Experiment2Locations {
17   repeated Experiment2_prop experiment2 = 1;
18 }
```



```
1 # -*- coding: utf-8 -*-
2 # Generated by the protocol buffer compiler. DO NOT EDIT!
3 # source: experiment2.proto
4 """Generated protocol buffer code."""
5 from google.protobuf import descriptor as _descriptor
6 from google.protobuf import message as _message
7 from google.protobuf import reflection as _reflection
8 from google.protobuf import symbol_database as _symbol_database
9 # @@protoc_insertion_point(imports)
10
11 _sym_db = _symbol_database.Default()
12
13
14
15
16 DESCRIPTOR = _descriptor.FileDescriptor(
17   name='experiment2.proto',
18   package='Experiment2Data',
19   syntax='proto2',
20   serialized_options=None,
```

Results – Experiment 1

Action	Time (seconds)	File Size (Mb)
Convert GPKG to GeoJSON using Pandas (<code>gpd.read_file()</code> and <code>gpd.to_file()</code>)	327s, std-dev 11.3s	288Mb (GPKG file) 614Mb* (GeoJSON file)
Load GeoJSON into Python using GeoPandas <code>gpd.read_file()</code>	81s, std-dev 3.2s	614Mb*
GeoJSON → Apache Avro (Serialize)	301s, std-dev 3.4s	228Mb
GeoJSON → Protocol Buffers PBF (Serialize)	306s, std-dev 2.9s	235Mb
Protocol Buffers PBF → GeoJSON (Deserialize)	378s, std-dev 2.8s	542Mb
Apache Avro → GeoJSON (Deserialize)	389s, std-dev 3.1s	546Mb

Results – Experiment 1a

Action	Time (seconds)	File Size (Mb)
Convert GPKG to GeoJSON using Pandas <code>gpd.read_file()</code> and <code>gpd.to_file()</code>	3.41s, std-dev 0.03s	5.1Mb (GPKG file) 8.1Mb (GeoJSON file)
Load GeoJSON into Python using GeoPandas <code>gpd.read_file()</code>	0.85s, std-dev 0.32s	8.1Mb*
GeoJSON → Apache Avro (Serialize)	3.17s, std-dev 0.05s	3.8Mb
GeoJSON → Protocol Buffers PBF (Serialize)	3.19s, std-dev 0.04s	3.8Mb
Protocol Buffers PBF → GeoJSON (Deserialize)	3.87s, std-dev 0.05s	6.9Mb
Apache Avro → GeoJSON (Deserialize)	3.98s, std-dev 0.03s	6.9Mb

Results – Experiment 2

Action	Time (seconds)	File Size (Mb)
JSON API response data download	variable	12.9Mb
JSON → GeoJSON	1.23s, std-dev 0.07s	11.5Mb*
JSON → Apache Avro (Serialize)	0.34s, std-dev 0.04s	7.0Mb
JSON → Protocol Buffers PBF (Serialize)	0.32s, std-dev 0.04s	7.1Mb
Protocol Buffers PBF → GeoJSON (Deserialize)	1.14s, std-dev 0.07s	11.5Mb
Apache Avro → GeoJSON (Deserialize)	1.10s, std-dev 0.03s	11.5Mb

* Note that the **encodingType** and **crs** fields are not implemented from the original JSON respons dataset.

Results Discussion (time, space)

- **Exp1** Binary files 20% smaller than GPKG, 63% smaller than GeoJSON
- **Exp1a** Binary files 26% smaller than GPKG, 54% smaller than GeoJSON
- **Exp2** Binary files 40% smaller than GeoJSON

- **Exp1** – no major timing differences observed
- **Exp1a** – similar to Exp1, no significant differences
- **Exp2** – Serialisation to Binary 3.6 times faster than serialization to GeoJSON

Results Discussion - Practicalities



- Binary files – **schemas always required** for (de)-serialization (+PROTOC class for Protobuf)
- **Apache Avro – no class compilation required**
- **Binary files will require specialist code generation for query/search** – many libraries provide this for JSON, GeoJSON, XML, etc...
- **Vendor lock-in avoided, good programming language support overall** – specialist knowledge required
- **Schemas will require updates if underlying data models change.** This could be problematic.

Real world Example: OpenStreetMap – dissemination of data in PBF format

download.geofabrik.de/europe/italy.html

Download OpenStreetMap data for this region:

Italy

[\[one level up\]](#)

The OpenStreetMap data files provided on this server do **not** contain the user names, user IDs and changeset IDs of the objects because these fields are assumed to contain personal information about the OpenStreetMap contributors and are therefore subject to data protection regulations in the European Union.

[Extracts with full metadata](#) are available to OpenStreetMap contributors only.

Commonly Used Formats

- [italy-latest.osm.pbf](#), suitable for Osmium, Osmosis, imposm, osm2pgsql, mkgmap, and others. This file was last modified 8 hours ago and contains all OSM data up to 2021-10-05T20:21:28Z. File size: 1.6 GB; MD5 sum: [22e37644df856809b7ec4e0c85139b41](#).
- [italy-latest-free.shp.zip](#) is not available for this region; try one of the sub-regions.

Other Formats and Auxiliary Files

- [italy-latest.osm.bz2](#), yields OSM XML when decompressed; use for programs that cannot process the .pbf format. This file was last modified 3 hours ago. File size: 2.7 GB; MD5 sum: [c9630291e333708d4584e996e2684fd4](#).
- [italy-internal.osh.pbf](#) The history file contains personal data and is available on the [internal server](#) only for further information.
- [.poly file](#) that describes the extent of this region.
- [.osc.gz files](#) that contain all changes in this region, suitable e.g. for Osmosis updates
- [raw directory index](#) allowing you to see and download older files

Sub Regions

Click on the region name to see the overview page for that region, or select one of the file extension links

Sub Region	Quick Links		
	.osm.pbf	.shp.zip	.osm.bz2
Centro	[.osm.pbf] (266 MB)	[.shp.zip]	[.osm.bz2]
Isole	[.osm.pbf] (147 MB)	[.shp.zip]	[.osm.bz2]
Nord-Est	[.osm.pbf] (493 MB)	[.shp.zip]	[.osm.bz2]
Nord-Ovest	[.osm.pbf] (433 MB)	[.shp.zip]	[.osm.bz2]
Sud	[.osm.pbf] (264 MB)	[.shp.zip]	[.osm.bz2]

imposm.org/docs/imposm.parser/latest/

[imposm.parser 1.0.7 documentation](#) »

imposm.parser - OpenStreetMap XML/PBF parser for Python

`imposm.parser` is a Python library that parses OpenStreetMap data in [XML](#) and [PBF](#) format.

It has a simple API and it is fast and easy to use. It also works across multiple CPU/cores for extra speed.

It is developed and supported by [Omniscale](#) and released under the [Apache Software License 2.0](#).

osmium.org/libosmium/

[HOME](#) [STATUS](#) [DOCUMENTATION](#) [CONTACT](#) [CODE](#)



Osmium Library

A fast and flexible C++ library for working with OpenStreetMap data

Features

The Osmium Library has extensive support for all types of OSM entities: nodes, ways, relations, and changesets. It allows reading from and writing to OSM files in XML and PBF formats, including change files and full history files. Osmium can store OSM data in memory and on disk in various formats and using various indexes. Its easy to use *handler* interface allows you to quickly write data filtering and conversion functions. Osmium can create WKT, WKB, OGR, GEOS and GeoJSON geometries for easy conversion into many GIS formats and it can assemble multipolygons from ways and relations.

wiki.openstreetmap.org/wiki/PBF_Formats


```
message Way {
  required int64 id = 1;
  // Parallel arrays.
  repeated uint32 keys = 2 [packed = true];
  repeated uint32 vals = 3 [packed = true];

  optional Info info = 4;

  repeated sint64 refs = 8 [packed = true]; // DELTA coded

  // The following two fields are optional. They are only used in a special
  // format where node locations are also added to the ways. This makes the
  // files larger, but allows creating way geometries directly.
  //
  // If this is used, you MUST set the optional_features tag "LocationsOnWays"
  // and the number of values in refs, lat, and lon MUST be the same.
  repeated sint64 lat = 9 [packed = true]; // DELTA coded, optional
  repeated sint64 lon = 10 [packed = true]; // DELTA coded, optional
}
```

Final thoughts ...



final
thoughts

- **It still remains a challenge to measure and understand “success”** in regards to the possible replacement of existing ‘de-facto’ standards with binary data serialization
- **Obvious and quantifiable performance advantages with binary data serialization**
- However, **overheads remains which could impede wider adoption** include – schema updating, specialist knowledge, small worldwide user community, etc.
- **More spatial or location-data specific experimentation required in future work.**



European
Commission

With many thanks for watching and listening

Peter Mooney (IE)

Email: peter.mooney@mu.ie



European Commission
Joint Research Center
JRC/B/06

EXPERT CONTRACT
CONTRACT NUMBER - CT-EX2014D166355-104