Agris on-line Papers in Economics and Informatics

Volume VI Number 1, 2014

# Spatial Data Monitoring and Mobile Applications – Comparison of Methods for Parsing JSON in Android Operating System

J. Masner, J. Vaněk, M. Stočes

Faculty of Economics and Management, Czech University of Life Sciences in Prague, Czech Republic

## Anotace

Příspěvek se na obecné úrovni zabývá problematikou zpracování formátu JSON pro zpracování v mobilních zařízeních na platformě operačního systému Android. Implementace je testována a demonstrována na zpracování rozsáhlé kolekce pozičních dat, která vznikají monitorováním pohybu zvěře (aplikace Zvěř online). Možností využití je obecně nejen v oblasti zemědělství, rozvoje venkova a ochraně životního prostředí prakticky neomezené množství, s narůstajícím počtem a dostupností mobilních zařízení nabývá řešená problematika dále na významu. Při vývoji aplikací pro mobilní zařízení je podstatné, jakým způsobem lze získávat aktuální data pro běh vlastní aplikace. Jelikož nelze přistupovat přímo k databázím, je nutné mezi klientem a serverem přenášet již vybraná data pomocí vhodného přenosového formátu. K tomuto účelu slouží serializace dat, kde jednou z používaných forem je formát JSON. Konkrétně jsou zde porovnávány metody pro jeho parsování, tedy převod do objektů, případně kolekcí, které se dají v aplikacích využít pro další zpracování. Rozhodující je zde především časové hledisko, které představuje dobu trvání zejména samotného převodu.

## Klíčová slova

Android, JSON, data serialization, InputStream, XML, Protocol Buffers, Java, libraries, Game Online.

## Abstract

The paper generally addresses the issue of processing the JSON format for mobile devices on the Android operating system platform. Implementation has been tested and demonstrated using the processing of an extensive collection of spatial data generated from game movement monitoring (the Game Online application). The potential for use not only in the sphere of agriculture, rural development and environmental protection is in general practically unlimited, and with the growing number and availability of mobile devices the discussed issue gains further relevance. The ways of obtaining current data for the operation of the application are essential for the development of applications intended for mobile devices. As direct access to databases is not possible, it is necessary to transfer preselected data between the client and server using a suitable transfer format. This is catered for by data serialization, where one of the applied forms is the JSON format. Specifically, the paper compares methods of its parsing, i.e. transfer to objects, or, where applicable, collections, which may be utilised in the applications for further processing. The decisive factor here is namely the time which represents, in particular, the duration of the transfer proper.

## Key words

Android, JSON, data serialization, InputStream, XML, Protocol Buffers, Java, spatial data, Game Online.

## Introduction

Recently, mobile devices have become an integral part of life of all people throughout the civilised world. The sales figures for tablets outnumber those of PCs – especially laptops. The number of smartphones sold in 2013 for the first time exceeded the number of classical mobile phones (Gartner, 2014). According to the same study, 78% of sold devices uses the Android operating system. According to (StatCounter, 2014) as well as a number of other studies and statistics, in 2013 Android ranked as the first operating system among those deployed in the used devices.

When developing an application, it is often

necessary to bear in mind also communication with the surroundings – particularly the Internet environment, which means that applications are not isolated units. It is necessary to gain current data and information. Furthermore, applications for mobile devices are often a required and needed complement to existing web applications. Data for mobile devices are updated via the Internet. For security reasons it is not possible to connect directly to database systems, which would be probably the most rapid solution. Instead, applications connect to web servers, which mediate the selection of data from these systems. For this reason it is necessary to select the method of data serialization in the transfer between the application (client) and the server.

In existing applications, primarily with a view to the extensive competition, much attention has to be paid to the performance and the speed of displaying and processing of information. Although the performance of mobile devices keeps growing, software is becoming more and more complex. Mobile device users have many applications installed, and many of them utilise services at the background of the system, which run constantly or are triggered at certain time intervals.

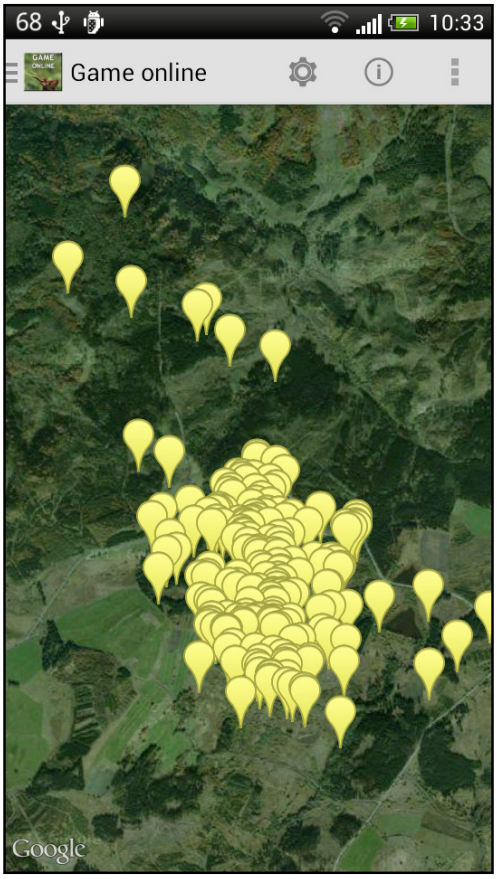
When selecting the method of serialization it is necessary to consider several aspects. One of them is the potential and speed of processing on the part of the server. Another one is the volume of data in the given format for communication. This affects primarily the speed of transfer between the client and the server. Furthermore, mobile devices are often connected via mobile operator networks, where the decisive economic factor is the volume of data. Moreover, especially in rural regions signal coverage as well as the technologies used are often of a significantly lower standard than those applied in cities (persisting technological digital divide). Another aspect is also the speed of processing (parsing) in client devices.

Specialised (map, spatial, etc.) applications often require a transfer and processing of large quantities of data. The primary information in applications using spatial data which are displayed on map data are the coordinates of individual positions. These often have to be extended with a whole set of other additional information, frequently of large volume. The paper discusses the issue of spatial data processing in the monitoring of game movements in a mobile application for the Android operating system, specifically the conversion of data transferred between the server and the client (in the server → client direction) and their conversion to objects and collections for further processing in the mobile application. The general objective has been to compare the methods of parsing the JSON format in terms of time demands and the number of rows/objects which is utilised here as well as in many other solutions.

## Materials and methods

Within the scope of research conducted at the Department of Information Technologies of the Faculty of Economics and Management of the Czech University of Life Sciences Prague, in cooperation with the Faculty of Forestry and Wood Sciences, and Vojenské lesy a statky, and others, the Game Online web application has been developed (available from **zver.agris.cz**). It is based upon the monitoring of game movement using special collars. Subsequently, an application for mobile devices with the Android operating system, which currently represents the most wide-spread system environment of mobile devices, is being developed.

The designated animals are followed up via GPS, by means of a collar (Fig. 3), which records the position of the animal with the accuracy of several meters; the location; records of the animal‘s GPC receiver; the date and time in programmed intervals (i.e. usually 1 hour). The collar also contains an activity sensor which records the animal‘s activity (such as feeding, resting, or movement) (Owen-Smith, 2012). In addition, the sensor records the temperature and the accuracy of measurements. Newer collars are equipped with GSM modules which contain telephone SIM (*subscriber identity module*) cards allowing for the transfer of data to the user‘s computer. Generally, it is useful to validate the data and to store them in a database system for subsequent processing, display. (Fan, 2004) Information on the movement of wild animals is obtained by means of the GPS (*Global Positioning System*) in the collar equipped with a GSM module. Data are received via a ground station, thereafter validated and stored in a database server. The selection of data from the database is then safeguarded by the Game Online web application. This processes the data for mobile device applications and provides them in the necessary format (XML, JSON). The resulting application is shown in Fig. 1.




Source: own processing

Figure 1: Game application sample.

For the Android operating system, programming is performed using special development tools - *Eclipse ADT* or *AndroidStudio*, and the *Java* programming language. Language translation is then carried out by the Android's own virtual machine *Dalvik*, (from version 4.4 OS Kitkat it has newly been ART) (Android, 2014), rather than by the classical method to bytecode for Java Virtual machine (source).

The XML and JSON technologies are most often used for communication between the applications (clients) and the server. Another technology which could be utilised is the Protocol Buffers.

*Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.* (Google Developers, 2014) Nevertheless, due to the used technologies of the web application, the processing of Protocol Buffers is problematic on the part of the server.

XML (eXtensible Markup Language) is a markup language which is used in a number of applications as a data transfer or storage format. It is well readable and recordable both for people and machines. Its assets are the widespread usage and support of a number of development tools. Its minuses, on the contrary, are the higher demands for data volume – particularly compared to the other above-mentioned JSON technology. The difference in data volumes for tested data samples is provided in the table below.

| Number of positions | XML | JSON |
|---|---|---|
| 100 | 33KB | 23KB |
| 500 | 165KB | 114KB |
| 1000 | 329KB | 227KB |

Source: own processing.

Table 1 Comparison of data sizes for XML and JSON formats.

JSON (JavaScript Object Notation) is a lightweight data exchange format. It is based upon JavaScript. It utilises the convention known from C language-based programming languages (json.org, 2014). Like the XML markup language it is easily readable and recordable both for people and machines.

The JSON format is based upon two basic structures. One of them is the name value pair collection – in programming languages often called the object. It is enclosed in curly brackets {}. The value may be a string, number, object, field, true, false, or null (json.org, 2014). Another structure is a field. This is an ordered list of values, most often objects, which is enclosed in square brackets [].

Most authors discuss the comparison of parsing speed between the XML, JSON and, where applicable, Protocol Buffers technologies (Rodrigues, 2011) (Chen, 2013). Nevertheless, in the development for Android, several methods may be applied to the parsing of JSON alone. Each of them has a different duration. In terms of complexity, all of these methods exhibit a similar program code length, but utilise different libraries. Five available libraries were selected for the purposes of testing.

The first method, called Android, utilises parsing libraries supplied directly by Google as part

of the Android SDK (*Software Development Kit*) (org.json, 2014). Unlike with other methods, it is not necessary to compile additional libraries into the final application. The disadvantage is, however, that in communication with the server the downloaded data are in the *InputStream* class object, which requires conversion to a String class object. With large data volumes, this may be extremely demanding (Figure 2).

Another tested method, called *Gson*, comes from the *com.google.gson.stream.JsonReader* package. The google-gson Java library is also by Google and it is intended for the Java programming language (google-gson, 2014). The used version was 2.2.4. (Figure 3).

```
List<Deer> result = new ArrayList<Deer>();
try {
    String json = convertStreamToString(inputStream);
    JSONObject jsonObject = new JSONObject(json);
    Iterator iterator = jsonObject.keys();
    while (iterator.hasNext()) {
        List<HashMap<String, String>> positions = new ArrayList<HashMap<String,String>>();
        String idKey = (String) iterator.next();
        String id = jsonObject.get(idKey).toString();
        JSONArray deerArray = jsonObject.getJSONArray(idKey);
        int length = deerArray.length();
        for (int i = 0; i < length; i++) {
            HashMap<String, String> pos = new HashMap<String,String>();
            JSONObject posObject = deerArray.getJSONObject(i);
            Iterator posIterator = posObject.keys();
            while(posIterator.hasNext()) {
                String key = (String) posIterator.next();
                String value = (String) posObject.get(key);
                pos.put(key, value);
            }
            positions.add(pos);
        }
        result.add(new Deer(id,positions));
    }
```

Source: own processing.

Figure 2: Sample parsing using the AndroidJson method - org.json.

```
String id;
JsonReader reader = new JsonReader(new InputStreamReader(inputStream, "UTF-8"));
reader.beginObject();
while (reader.hasNext()) {
    id = reader.nextName();
    reader.beginArray();
    List<HashMap<String, String>> positions = new ArrayList<HashMap<String,String>>();
    while (reader.hasNext()) {
        reader.beginObject();
        HashMap<String, String> pos = new HashMap<String,String>();
        while (reader.hasNext()){
            String key = reader.nextName();
            String value = reader.nextString();
            pos.put(key, value);
        }
        positions.add(pos);
        reader.endObject();
    }
    reader.endArray();
    result.add(new Deer(id,positions));
}
reader.endObject();
reader.close();
```

Source: own processing.

Figure 3: Sample parsing using the Gson method - org.json com.google.gson.stream.JsonReader.

Another method, JSON.simple, uses the JSON. simple library (json-simple, 2014). The used version was 1.1. Parsing uses the heap based method (Figure 4).

Another used method, JSON.smart, is, in terms of the program code, almost identical to JSON. simple. It is performance-driven (json-smart, 2014). The used version was 1.1.1. (Figure 5).

```
JSONParser p = new JSONParser();
String id;
Set keys;
JSONObject jsonObject = (JSONObject) p.parse(new InputStreamReader(inputStream));
keys = jsonObject.keySet();
for (Object key: keys) {
    id = (String) key;
    List<HashMap<String, String>> positions = new ArrayList<HashMap<String,String>>();
    JSONArray jsonArray = (JSONArray) jsonObject.get(key);
    int size = jsonArray.size();
    for (int i = 0; i < size; i++) {
        HashMap<String, String> pos = new HashMap<String,String>();
        JSONObject positionJsonObject = (JSONObject) jsonArray.get(i);
        Set posKeys = positionJsonObject.keySet();
        for (Object posKey:posKeys) {
            String hashKey = (String) posKey;
            String hashValue = positionJsonObject.get(posKey).toString();
            pos.put(hashKey, hashValue);
        }
        positions.add(pos);
    }
    result.add(new Deer(id,positions));
}
```

Source: own processing.

Figure 4: Sample parsing using the JSON.simple method - org.json.simple.

```
JSONParser p = new JSONParser(JSONParser.MODE_JSON_SIMPLE);
String id;
Set keys;
JSONObject jsonObject = (JSONObject) p.parse(inputStream);
keys = jsonObject.keySet();
for (Object key: keys) {
    id = (String) key;
    List<HashMap<String, String>> positions = new ArrayList<HashMap<String,String>>();
    JSONArray jsonArray = (JSONArray) jsonObject.get(key);
    int size = jsonArray.size();
    for (int i = 0; i < size; i++) {
        HashMap<String, String> pos = new HashMap<String,String>();
        JSONObject positionJsonObject = (JSONObject) jsonArray.get(i);
        Set posKeys = positionJsonObject.keySet();
        for (Object posKey:posKeys) {
            String hashKey = (String) posKey;
            String hashValue = positionJsonObject.get(posKey).toString();
            pos.put(hashKey, hashValue);
        }
        positions.add(pos);
    }
    result.add(new Deer(id,positions));
}
```

Source: own processing.

Figure 5: Sample parsing using the JSON.smart method - net.minidev.json

```
JSONParser p = new JSONParser(JSONParser.MODE_JSON_SIMPLE);
String id;
Set keys;
JSONObject jsonObject = (JSONObject) p.parse(inputStream);
keys = jsonObject.keySet();
for (Object key: keys) {
    id = (String) key;
    List<HashMap<String, String>> positions = new ArrayList<HashMap<String,String>>();
    JSONArray jsonArray = (JSONArray) jsonObject.get(key);
    int size = jsonArray.size();
    for (int i = 0; i < size; i++) {
        HashMap<String, String> pos = new HashMap<String,String>();
        JSONObject positionJsonObject = (JSONObject) jsonArray.get(i);
        Set posKeys = positionJsonObject.keySet();
        for (Object posKey:posKeys) {
            String hashKey = (String) posKey;
            String hashValue = positionJsonObject.get(posKey).toString();
            pos.put(hashKey, hashValue);
        }
        positions.add(pos);
    }
    result.add(new Deer(id,positions));
}
```

Source: own processing.

Figure 6: Sample parsing using the Jackson method - com.fasterxml.jackson.core.

The last method used, called Jackson, uses the Jackson JSON Processor library. The library has been inspired by tools for XML – StAX, JAXB, etc. (JacksonHome, 2014). The used version was 2.3. (Figure 6).

In parsing, all of the methods employ browsing of individual structures using iterations in a similar manner. The samples shown above illustrate that the complexity of their source codes is similar. Therefore, in terms of implementation complexity, they are on a very similar level. The only exception is the Android method, which requires also the creation of a function for the conversion of an InputStream class object to a String one.

In order to achieve the established objective, a test application has been developed which gradually triggers all of the aforementioned methods. Parsing itself was in all cases closed in the try {}catch{} clause for the identification of exceptions (errors). Each method was enclosed in its own object with the needed methods. The time necessary for parsing using the respective method was measured always prior to triggering the method of the instance which safeguards parsing proper. Below, the measured time is specified in milliseconds.

```
long duration = 0;
long start = System.currentTimeMillis();
testJson.parsePublicTimeline(inputStream);
duration += (System.currentTimeMillis() - start);
```

Source: own processing.

Picture 7: Sample of parsing measurements proper

Data were converted with a view to the Game Online target application into a collection of objects dedicated to this purpose. This object represents one individual (animal) and all of the positions associated therewith. It has one ID attribute, and also contains a collection of HashMap <String, String> class objects for the positions.

```
long duration = 0;
long start = System.currentTimeMillis();
testJson.parsePublicTimeline(inputStream);
duration += (System.currentTimeMillis() - start);
```

Source: own processing.

Picture 7: Sample of parsing measurements proper

Measurements were performed gradually for 10, 50, 100, 200, 500, 1000, and 5000 spatial data. On the highest level, the tested JSON contains an object representing individual animals broken down by their IDs. A field of objects representing individual positions is then allocated to each individual.


```
{
  "12110":[
     {
        "specie_name":"Jelen Evropsk\u00fd",
        "name":"Barka",
        "color":"00FFFF",
        "id_deer":12110,
        "UTC_DATE":"21.01.2014",
        "UTC_TIME":"22:00:44",
        "LMT_TIME":"23:00:44",
        "TEMP":7,
        "LATITUDE":"50,303611",
        "LONGITUDE":"13,0705745",
        "RowNum":"1"
     }
  ],
  "12268":[
     {
        "specie_name":"Jelen Evropsk\u00fd",
        "name":"Bonif\u00e1c",
        "color":"7fffd4",
        "id_deer":12268,
        "UTC_DATE":"21.01.2014",
        "UTC_TIME":"22:00:43",
        "LMT_TIME":"23:00:43",
        "TEMP":7,
        "LATITUDE":"50,6010005",
        "LONGITUDE":"13,3487443",
        "RowNum":"1"
     }
  ]
}
```


Source: own processing.

Figure 9: Simplified sample of tested data - JSON.

In the course of testing, no statistically significant difference was observed in the design with one tested individual, or with more tested individuals. Therefore the testing of one individual was used in all cases.

The measurements were carried out gradually on 3 mobile devices with the Android operating environment:

1. Samsung Galaxy Note 10.1 (GT-N8010)
   - ARM Cortex A9 quad-core processor with 1.4 Ghz frequency
   - 2GB operational memory
   - OS Android version 4.1.2 – API level 16
2. HTC One S
   - Qualcomm MSM8260 1.7 Ghz dual-core processor
   - 1GB operational memory
   - OS Android 4.1.1 – API level 16
3. Samsung Galaxy Mini
   - Qualcomm MSM7227 600 Mhz single-core processor
   - 512 MB operational memory
   - OS Android 2.3 – API level 9

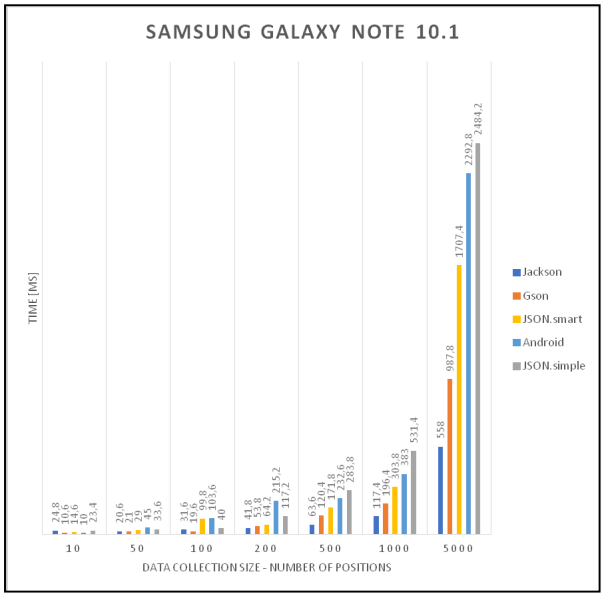
The *airplane mode* was set up on the devices, which switched off any networks and synchronisations in order to avoid any adverse external influences and to terminate maximum applications, i.e. for the processor time during parsing to be employed by other processes. Background services in particular, and other scheduled tasks, could adversely affect the result of measurements. JSON for each tested data scope was stored in *Assets*. From these, an InputStream class object may be obtained easily, which would otherwise be obtained during communication with the server. This allowed for the disconnection of the device from any networks.

## Results and discussion

Graphs 1 - 3 provide the measured times necessary for parsing on individual devices. All of the values are provided in ms units. The measuring proper (triggering the application) was conducted ten times for each device. Thereafter, averages were obtained from the measured values in order to minimise adverse influences.

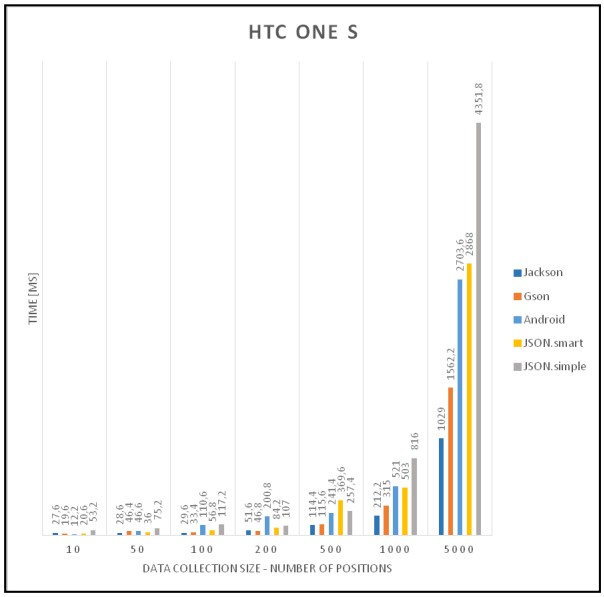
It is quite obvious that for large data collections the use of the Jackson method (*Jackson JSON Processor* library) is the most advantageous one. Results indicate that its use is best from approximately 50 positions. This method, however, is not suitable for small collections, where it actually proves to be one of the worst. Good results in this sphere were achieved also by the *Gson* method used with a single-core processor. For multi-core processors, the best one seems to be the *Android* method. Its results, however, are extremely poor for large data collections, although the method is supplied as part of Android SDK.

The very good results of the *Android* method if used for small data collections could be probably explained by the use of multi-core processors. This method is part of native SDK and is probably able to cooperate better with the operating system and to use the hardware, particularly the possibility to run in several threads. With the growing size of the data collection, the time necessary for the conversion of an *InputStream* class object to *String* seems to play a decisive role in this respect. This fact is further supported



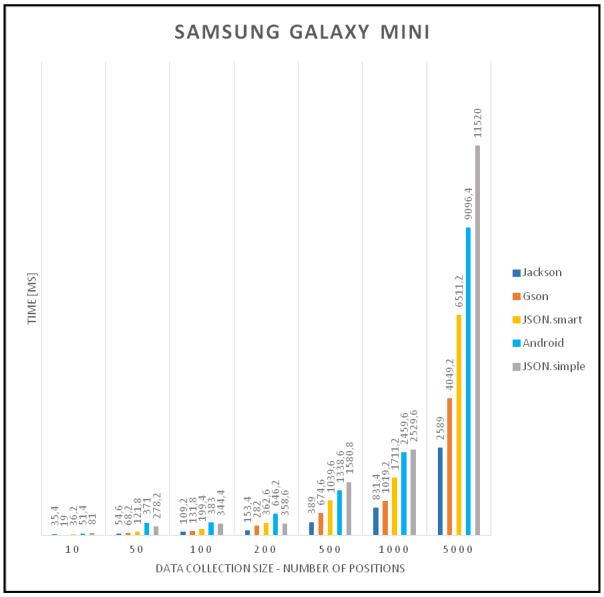

Source: own processing

Graph1: Measured values for Samsung Galaxy Note 10.1.




Source: own processing

Graph 2: Measured values for HTC One S.




Source: own processing

Graph 3: Measured values for Samsung Galaxy Mini.

by the results of the single-core processor where this method ranges among the worst ones starting from the smallest data collection. This fact is generally worth a more detailed examination. There is an option to measure the time for this method only after the conversion to *String* proper. Nevertheless, when obtaining application data from the server, the obtained data are in the *InputStream* class object and it will always require conversion to *String*.

An exact determination of the data collection size from which it is optimal to use the Jackson method would require a more detailed study in the range around the 50-position size. The obtained results, however, are sufficient for the objective of the research. In general, the scope of data used by the application should be taken

into consideration. Determination of the data collection size and subsequent program branching by results would probably lead to an unacceptable time delay which would be longer than the use of a non-optimal method.

## Conclusion

The purpose of the research was to identify an optimal method for parsing the JSON format for use in the sphere of spatial data. With a view to the results of measurements, the *Jackson JSON Processor* library may be recommended for use in such applications. Nevertheless, it is necessary to take into account the amount of positions to be displayed by the given application. If it concerned small number at all times, it would be optimal to use the native library, or, if applicable, google-gson. In applications for agriculture, the environment, etc. as well as in the usage of applications for further research, it is necessary to display large quantities of positions. The objective of the conducted research was thus clearly met.

The presented results provide a basic view of the discussed issue. To be able to draw general conclusions about the optimal method for JSON format parsing, it would be certainly appropriate to conduct measurements with more devices. Moreover, it would be interesting to carry out the measurements for various data structures and various sizes of the individual values of the JSON structure. For the purposes of objectives set for this paper, however, the completed measurements are adequate. Further possibilities and procedures outlined will be the subject of follow-up research.

In further research which would enable to draw generally applicable conclusions it would be interesting to continue the testing also with various types of data. Furthermore, it would be appropriate to compare other possible formats for data transfer and serialization, such as XML, Protocol Buffers, etc., for the purposes of the discussed utilisation.

Application of other approaches to the design of similar applications would be also worth consideration. If, in the course of time, new data are only added to the data base and the existing ones remain unchanged, there is a possibility to store the data on an ongoing basis in the device database, although this implies a number of other potential problems. The database could take up too much memory in the device. It would also be necessary to safeguard data updates which could be, depending on the application usage, more time demanding than a simple download of the currently needed data. The implementation of such solution would be generally more complex. Economic efficiency could pose another problem.

## Acknowledgments

Pieces of knowledge introduced in this paper resulted from solution of an institutional research intention. University Internal Grant Agency of the Czech University of Life Sciences in Prague, grant no. 20134312, “Analysis and visualization of GPS telemetry outputs of cloven-hoofed animals in Doupov Mountains and the Bohemian Switzerland National Park”. The discussed issue also contributes to further direction of research, specifically then to the theoretical part of the prepared grant of the Internal Grant Agency of the Faculty of Economics and Management of the Czech University of Life Sciences in Prague, grant no. 20141048, “Processing of large data collections in JavaScript map API for www environment”.

*Corresponding author:*
*Ing. Jan Masner*
*Department of Information Technologies, Faculty of Economics and Management,*
*Czech University of Life Sciences in Prague, Kamýcká 129, 165 21 Prague 6 - Suchdol, Czech Republic*
*E-mail: masner@pef.czu.cz*

## References

[1] Rodrigues, C., José Afonso, J. , Tomé, P. Mobile Application Webservice Performance Analysis: Restful Services with JSON and XML. 2011. DOI: 10.1007/978-3-642-24355-4_17.

[2] Chen, H., Liou, Y., Chen, S., Li, J. Design of mobile healthcare service with health records format evaluation. 2013. DOI: 10.1109/ISCE.2013.6570215.

[3] Jackson JSON Processor Wiki. JacksonHome [online]. [cit. 2014-02-23]. Available: http://wiki.fasterxml.com/JacksonHome.

[4] Json-smart: A small and very fast json parser/generator for java. Google Code [online]. [cit. 2014-02-23]. Available: https://code.google.com/p/json-smart/.

[5] Json-simple: JSON.simple - A simple Java toolkit for JSON. Google Code [online]. [cit. 2014-02-23]. Available: https://code.google.com/p/json-simple/ .

[6] Google-gson: A Java library to convert JSON to Java objects and vice-versa. Google Code [online]. [cit. 2014-02-23]. Available: https://code.google.com/p/google-gson/ .

[7] Org.json: Google. GOOGLE. Android Developers [online]. [cit. 2014-02-23]. Available: http://developer.android.com/reference/org/json/package-summary.html.

[8] Úvod do JSON. JSON.ORG. [online]. [cit. 2014-02-23]. Available: http://www.json.org/json-cz.html.

[9] Protocol Buffers: What Are Protocol Buffers?. GOOGLE. Google Developers [online]. April 2, 2012 [cit. 2014-02-23]. Available: https://developers.google.com/protocol-buffers/.

[10] StatCounter Global stats: Top 8 Mobile Operating Systems. STATCOUNTER. [online]. [cit. 2014-02-23]. Available: http://gs.statcounter.com/#mobile_os-ww-monthly-201301-201401.

[11] Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013. GARTNER. [online]. 2014 [cit. 2014-02-23]. Available: http://www.gartner.com/newsroom/id/2665715

[12] Introducing ART. GOOGLE. Android [online]. [cit. 2014-02-23]. Available: https://source.android.com/devices/tech/dalvik/index.html.

[13] Owen-Smith, N., Goodall, V., Fatti, P. Applying mixture models to derive activity states of large herbivores from movement rates obtained using GPS telemetry. Wildlife Research, 2012, 39 (5), p. 452-462. ISSN 1035-3712.

[14] Fan, F., Biagioni, E. S. An approach to data visualization and interpretation for sensor networks, Paper presented at the Proceedings of the Hawaii International Conference on System Sciences, 2004, 37 999-1008.