The GPJSON Format Specification

——Json Format for Geophysics Specification

Abstract: GPJSON which derived from GeoJSON is only for encoding geophysical sensor source structures based on JavaScript Object Notation (JSON). It defines several types of JSON objects and the manner in which they are combined to represent a tree-table structure (the tree structure represent base information and table structure present sensor source in geophysical field acquisition and process). It also describe how to represent a concrete binary metadata (in string format in json) or link external file (local or URL) or pack matrix data by defining some special types. The goal of GPJSON is to integrate the geoscience source that are compatible with existing GeoJSON format and to parse and transmit source data from client (remote acquisition) to server (processing and interpretation) easily.

Note: Geoarch is a geoscience source processing platform focusing on Source-flow diagram schemes.

Memo

Date	Version	Authors	Email	Memo
14/02/2015	0.1	Aben Lee	aben.lee@foxmail.com	
05/05/2015	0.2	Aben Lee	aben.lee@foxmail.com	
27/07/2017	0.3	Aben Lee	aben.lee@foxmail.com	
15/03/2020	1.0	Aben Lee	aben.lee@foxmail.com	Modified base on GeoJson
24/03/2020	1.1	Aben Lee	aben.lee@foxmail.com	Modified by column to row
11/12/2022	2.0	Aben Lee	aben.lee@foxmail.com	Modified dataset to dataset and row column relation map similar to Echart API
11/12/2022	2.1	Aben Lee	aben.lee@foxmail.com	Adding 'bbox' and 'id' key-value for compatibility
11/12/2022	2.1	Aben Lee	aben.lee@foxmail.com	Adding Sequence' item for parse stream information IoT application

TODO: add code member for processing source data by using language interpreter

Table of Contents

1 Introduction

Because GeoJSON [RFC7946] is only a geospatial source interchange format based on JavaScript Object Notation (JSON) [RFC7159], GPJSON define several types of JSON objects the manner in which they are combined to represent geophysical sensor source mainly described as a combination of Tree - Table structure. In addition, GPJSON also define a several types that make a relevant GPJSON object represent binary source or link external file or pack matrix data. For example, the source GPJSON object, which is array's array of json formation, is a netCDF (network properties source format) metadata according concrete type which have been define in this document with 'binary' type, or is a URL directory of external file (*.nc) path with 'URI' type of DataSet object .

Like GeoJSON, GPJSON also present a feature collection of individual features. Each Feature has, at least 3 'attributes': a fixed value 'type' ('type': 'Feature'), a 'dataset' ,which is similar to 'geometry' of GeoJSON, but not for geospatial source, and a 'properties'. The dataset object SHOULD have 'type': '...'refer to source significance ('source': [[...]] which is array's array for the represent source or external file.

The GPJSON is mainly concerned with geoscience sensor source in the broadest fields, which mainly goal is to make geophysical sensors source parse and transmit easily from client (remote acquisition) to server (processing and interpretation) consistently via internet of Things. GPJSON can be used independently, it also compatible with GeoJSON format because of deriving from GeoJSON.

1.1 Requirements Language

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in [RFC2119].

1.2 Conventions Used in This Document

JSON (JavaScript Object Notation) is a lightweight, text-based, language-independent source interchange format. It includes 4 basic source types and 2 composite source types, and a total of 6 source types. The ordering of the members of any JSON object defined in this document MUST be considered irrelevant, as specified by [RFC7159].

In the following sections, JSON source types are represented as JSON+ space + source types, such as JSON Array.

Some examples use the combination of a JavaScript single-line comment (//) followed by an ellipsis (...) as placeholder notation for content deemed irrelevant by the authors. These placeholders must of course be deleted or otherwise replaced, before attempting to validate the corresponding JSON code example.

Whitespace is used in the examples inside this document to help illustrate the source structures, but it is not required. Unquoted whitespace is not significant in JSON.

1.3 Definitions

JavaScript Object Notation (JSON), and the terms object, member, name, value, array, number, true, false, and null, are to be interpreted as defined in [RFC7159].

Inside this document, the terms follow the definitions of [RFC7946]. There are specify additional term as following: type, id(option), features, properties, dataset, FeatureCollection, Sheet, Binary and URI, Bbox, Geometry etc..

For Tree-Table structure, the dataset should be parsed as sheet structure if type is 'Sheet' and the properties should be parsed as tree structure.

1.4 Example

A GPJSON FeatureCollection:

```
{
        'type': 'FeatureCollection',
        'id' : 1234，    // nember or string(option)
        'properties': {
                'prop0': 'value0'
         }
        'features': [{
            'type': 'Feature',
            'dataset': {
                'type': 'Sheet',   // the following json array should be parsed row table structure
                 'attributes' : ['Time/second', 'X/meter', 'Y/meter' ], // describe source columns attributes
                'source' : [['10.0', 12.4, 12.4],
                         ['30.0', 23.4,23.4],
                         ['50.0,34.7,34.7 ]] // the source of table
            },
            'properties': {    // should be parsed as tree structure
                'prop0': 'value0'
            }
          },{
            'dataset': {
                'type': 'Sheet',   // the following json array should be parsed row table source structure
                'attributes' : ['Time/second', 'X/meter', 'Y/meter' ],
                    'source' : [ {'Time/second':'10.0', 'X/meter' : 12.4, 'Y/meter' : 12.4},
                              [{'Time/second': '30.0', 'X/meter' :23.4, 'Y/meter' :23.4],
                              [{'Time/second': '50.0, 'X/meter' :34.7, 'Y/meter' :34.7 }]
            },
            'properties': {     // should be parsed as tree structure
                    'prop0': 'value0'
            }
          {
            'type': 'Feature',
            'dataset ': {
                'type': 'Binary',        // represent the source is a binary format
                'source':    [[....]]      // '...' is binary source. see Section 4.4 of [RFC7493].
            },
            'properties': {
                    'prop0': 'value0'
            }
          },{
            'type': 'Feature',
            'dataset ': {
                'type': 'URI',      // see Section 5 of [RFC4648]
                'source': [['https://192.10.0.186/GeoArch/manual']],      //'... ' is URL file in web serves,
            },
            'properties': {
                    'prop0': 'value0'
            }
        }]
}
```

Note: *The GPJSON texts should follow the constraints of Internet JSON (I-JSON) [RFC7493] for maximum interoperability.*

2 GPJSON object and text

GPJSON is derived from GeoJSON specification, that mean GPJSON object represents a Geometry, Feature, or collection of Features, see the Section 3 of see Section 2 & 3 of [RFC7493]. However, there are a special dataset object for describing geophysical sensor source information etc.

2.1 The properties of Feature

The properties object of Feature in GPJSON SHOULD be parsed a tree structure that is essential information for understanding the dataset. Including source description and creator and creating date etc.

The Feature also optional have a member geometry which have been defined in GeoJSON specification document.

The Featur have a crucial dataset member in this document which will be illustrated in more detailed in the Section 3 of this specification.

2.2 The properties of FeatureCollect

A GeoJSON object with the type "FeatureCollection" is a FeatureCollection object. A FeatureCollection object has a member with the key name "features". The value of "features" is a JSON array. Each element of the array is a Feature object as defined in this specification. It is possible for this array to be empty.

Additional, the GPJSON also should have a key named properties which object is all features's common properties, or same information of all features.

3 Dataset

The dataset member of Feature is mainly for packing a variety of geophysical sensor sources which are different from Geospatial source (or are not convenient for GeoJSON representation). There are 4 types for representing source manner. Every dataset's value is a JSON object no matter where it occurs in a GPJSON text.

The 'type' value of a dataset MUST be one of 'Sheet', 'Matrix', 'Binary' , 'URI' and 'Sequence', similar to type's value of Geometry of GeoJSON.

3.1 Sheet

Mostly sensor source can be represented by table (or column sheet) structure. For type 'Sheet', the other value of dataset elements SHOULD be JSON array. The header of table ('attributes': […]) MAY be optional, which represents the column header information against source of dataset. Every header string MAY be split by using '/' separator. The first string is column name in accordance with source array's name, and the second string refers to this column units, e.g. Temperature/Celsius be split Temperature and Celsius for the column source value and its unit.

The source of dataset should be array of arrays or array of objects, as shown the following

2D table example (array of arrays)

```
'dataset' :{
   ' type': 'Sheet',
   'attributes': ['Time/second','X/meter',    'Y/meter',    'Temperature/Celsius'],
   'source': [
       [1.00 ,23.00 ,13.00,20.00],
       [2.00,23.00 ,14.00 ,19.90]
       ......
```

```
    ]
}
```

Alternative 2D table example (array of objects)

```
'dataset' : {
    ' type': 'Sheet',
    'attributes': ['Time/second','X/meter',    'Y/meter',    'Temperature/Celsius'],
    'source': [
        ['Time/second': 1.00 , 'X/meter':23.00 , 'Y/meter':13.00, 'Temperature/Celsius':20.00],
        ['Time/second':2.00 , 'X/meter': 23.00 , 'Y/meter':14.00, 'Temperature/Celsius':19.90]
        ……
    ]
}
```

The above source of dataset should be parsed as following:

Time/second	X/meter	Y/meter	Temperature/Celsius
1.00	23.00	13.00	20.00
2.00	23.00	14.00	19.90
3.00	23.00	15.00	19.80
4.00	23.00	16.00	19.70
5.00	23.00	17.00	19.60
6.00	23.00	18.00	19.50
7.00	23.00	19.00	19.40
8.00	23.00	20.00	19.30
9.00	23.00	21.00	19.20
10.00	23.00	22.00	19.10

For example:

```
{
    ' type': 'Sheet',
    'attributes': ['Time/second','X/meter',    'Y/meter',    'Temperature/Celsius'],
    'source': [
        [1.00 ,23.00 ,13.00,20.00],
        [2.00,23.00 ,14.00 ,19.90]
        ……
    ]
}
```

3.2 Matrix

If the type's value is matrix, the source object of dataset represent a matrix dataset. At this point, the source MUST be array of arrays, and the omit attributes object. A row of matrix is corresponding to an array of arrays, that mean matrix source data stored in row-major order.

3.3 Binary

For type 'Binary'(JSON pairs), and the source object wrap a binary string for representing the raw data, and the attributes object SHOULD be a string to identify the binary's description, such as filename of binary string, which is relevant to the binary source wrapping string format.

For example:

```
{
   'type': 'Feature',
   'dataset': {
      'type': 'Binary',
      'attributes': ['abc.nc'],  // the attribute regard as fillname with suffix.
      'source': [['…']]          // the raw data of filename above.
   },
   'properties': {
       'prop0': 'value0'
   }
 }
```

NOTE: *binary string refer to Section 4.4 of [RFC7493].*

The ['abc.nc'] is the identify filename attributes to source’s data, and [['…']] is raw binary data which relevant to attributes. Since the identify filename is suffix with'.nc', that mean the [['…']] raw string is a NetCDF encoding formation.

However, it is RECOMMENDED that this source be encoded in a string value in base64url just as Section 4.3.

3.4 URI

For type 'URI'(JSON pairs), the source object SHOULD provide information,including file location and identify file name, to link an external file following the constraints of Internet JSON (I-JSON) [RFC7493]. The identify file name is the relevant to Uniform Resource Name, (URN), and the location information of identify file name also is relevant to Uniform Resource Locator(URL).

For example:

```
{
   'type': 'Feature',
   'dataset': {
      'type': 'URI',
      'source': [['c://source/abc.nc' ]]  //may be a web location.
   },
   'properties': {
       'prop0': 'value0'
   }
 }
```

The 'abc.nc' is the filename, and 'c://source' is the filename stored location. So, the '../source/abc.nc' is identified the file path and name.

Also, the source may be a file from a web serves, such as [[‘https:// 192.192.168.0/abc.nc’]] for link the address of the source data.

3.5 Sequence

For type 'Sequence'(JSON pairs), the source object SHOULD provide NULL information which mean the source may be an data stream of IoT following the Feature Object, and the data stream MUST be begin with string “SourceBegin” and then sequence array represented the source value of dataset above, and the data stream certainly MUST be end with string “SourceEnd”. the example as following.

For example1:

```
{
   'type': 'Feature',
   'properties': {
       'prop0': 'value0'
   }
   'dataset': {
      'type': 'Sequence',
      'attributes': ['Time/second',    'X/meter',    'Y/meter',    'Temperature/Celsius'],
      'source': NULL  //may be a link to location.
   },
},
'sourcebegin'  [...],[...]...[...], 'sourceend'
```

Because the data streams are Sequences, the are another example as following:

```
{
   'type': 'Feature',
   'properties': {
       'prop0': 'value0'
   }
   'dataset': {
      'type': 'Sequence',
      'attributes': ['Time/second',    'X/meter',    'Y/meter',    'Temperature/Celsius'],
      'source': NULL  //may be a link to location.
   },
},
'sourcebegin', {1: [...]}, {2:[...]}, .... {999:[...]},'sourceend'
```

Note: the sequence of type is only facilitates transmission of IoT information. And in most cases, the sequence of data stream should be encode cbor (Constrain Binary Object Representation) format for saving transmission bandwidth .

When transferring geophysics field data with embedded acquisition terminal, the embedded system have to transmits collecting information in time since its limited resources such as memory etc.. which mean embedded acquiring system can’t be waiting and wrapping the whole collecting data for transmitting it to the sever.

4. Date & Time

The Date & Time is not a JSON source type. For date types, we MUST use JSON strings. To make it easier for dates to be displayed and parsed, we SHOULD use a more internet-friendly format for dates in ISO 8601 format [rfc3339].

Example:

```
{       'created': '2018-07-12T16:20:57-05:00',
       'modified': '2018-07-12T16:20:57-05:00',}
```

5. Stamp

The stamp of properties can contain OPTIONAL attributes that may trace back the lifecycle of source. They are separated by ';' for different node processing information, and the information of each node can by split by '@' for look at node's name and generating time.

For example

```
      properties :{
      'stamp': '2018-07-12T16:20:57-05:00@filter_10; 2018-07-12T16:21:57-03:20@fit_13 ',
}
```

The lifecycle can be tracked back 2 nodes (filter_10 and fit_13) to processing the source data, and the created times are 2018-07-12T16:20:57-05:00 and 2018-07-12T16:21:57-05:00 respectively.

References:

http://GeoJson.org
NETCDF format specification
[RFC 2119] 'Key words for use in RFCs to Indicate Requirement Levels'
[RFC 4627] 'The application/json Media Type for JavaScript Object Notation (JSON)'
[RFC 2616] 'Hypertext Transfer Protocol'
[RFC 3339] 'Date and Time on the Internet: Timestamps'
Dixson, N., Milliken, G., Mukunda, K., Murray, R., & Starry, R. (2020). GeoJSON Source Curation Primer.
Joan Maso and Alaitz Zabala. (2017) Testbed-12 JSON and GeoJSON User Guide. https://docs.opengeospatial.org/guides/16-122r1.html.
[RFC 7493] The I-JSON Message Format.
[RFC 7946] The GeoJSON Format
https://echarts.apache.org/en/tutorial.html#Dataset