GeoJSON
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2016

H. Butler
Hobu Inc.
M. Daly
Cadcorp
A. Doyle
MIT
S. Gillies
Mapbox Inc.
T. Schaub
Planet Labs
S. Hagen

March 06, 2016

The GeoJSON Format
draft-ietf-geojson-01

Abstract

GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON). It defines several types of JSON objects and the manner in which they are combined to represent data about geographic features, their properties, and their spatial extents. This document recommends a single coordinate reference system based on WGS 84. Other coordinate reference systems are not recommended.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 7, 2016.

Table of Contents

1. Introduction

GeoJSON is a format for encoding a variety of geographic data structures using JavaScript Object Notation (JSON) [RFC7159]. A GeoJSON object may represent a region of space (a Geometry), a spatially-bounded entity (a Feature), or a list of features (a Feature Collection). GeoJSON supports the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection. Features in GeoJSON contain a geometry object and additional properties, and a Feature Collection contains a list of features.

The format is concerned with geographic data in the broadest sense; any thing with qualities that are bounded in geographical space might be a feature whether it is a physical structure or not. The concepts in GeoJSON are not new; they are derived from pre-existing open geographic information system standards and have been streamlined to better suit web application development using JSON.

GeoJSON comprises the seven concrete geometry types defined in the OpenGIS Simple Features Implementation Specification for SQL [SFSQL]: 0-dimensional Point and MultiPoint; 1-dimensional curve LineString and MultiLineString; 2-dimensional surface Polygon and MultiPolygon;

and the heterogeneous GeometryCollection. GeoJSON representations of instances of these geometry types are analogous to the well-known binary (WKB) and text (WKT) representations described in that same specification.

GeoJSON also comprises the types Feature and FeatureCollection. Feature objects in GeoJSON contain a geometry object with one of the above geometry types and additional members. A FeatureCollection object contains an array of feature objects. This structure is analogous to that of the Web Feature Service (WFS) response to GetFeatures requests specified in [WFSv1] or to a KML Folder of Placemarks [KMLv2.2]. Some implementations of the WFS specification also provide GeoJSON formatted responses to GetFeature requests, but there is no particular service model or feature type ontology implied in the GeoJSON format specification.

Since its initial publication in 2008 [GJ2008], the GeoJSON format specification has steadily grown in popularity. It is widely used in JavaScript web mapping libraries, JSON-based document databases, and web APIs.

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

The ordering of the members of any JSON object defined in this document MUST be considered irrelevant, as specified by [RFC7159].

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 data structures, but is not required. Unquoted whitespace is not significant in JSON.

1.3. Specification of GeoJSON

This document updates the original GeoJSON format specification [GJ2008].

1.4. 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 term "geometry type" refers to the seven case-sensitive strings: "Point", "MultiPoint", "LineString", "MultiLineString", "Polygon", "MultiPolygon", and "GeometryCollection".

- As another shorthand notation, the term "GeoJSON types" refers to the nine case-sensitive strings "Feature", "FeatureCollection" and the geometry types listed above.

- The word "Collection" in "FeatureCollection" and "GeometryCollection" does not have any significance for the semantics of array members. The "features" and "geometries" members, respectively, of these objects are standard ordered JSON arrays, not unordered sets.

1.5. Example

A GeoJSON feature collection:

```
{
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "geometry": {
            "type": "Point",
            "coordinates": [102.0, 0.5]
        },
        "properties": {
            "prop0": "value0"
        }
    }, {
        "type": "Feature",
        "geometry": {
            "type": "LineString",
            "coordinates": [
                [102.0, 0.0],
                [103.0, 1.0],
                [104.0, 0.0],
                [105.0, 1.0]
            ]
        },
        "properties": {
            "prop0": "value0",
            "prop1": 0.0
        }
    }, {
        "type": "Feature",
        "geometry": {
            "type": "Polygon",
            "coordinates": [
                [
                    [100.0, 0.0],
                    [101.0, 0.0],
                    [101.0, 1.0],
                    [100.0, 1.0],
                    [100.0, 0.0]
                ]
            ]
        },
        "properties": {
            "prop0": "value0",
            "prop1": {
                "this": "that"
            }
        }
    }]
}
```

2. GeoJSON Text

A GeoJSON text is a JSON text and consists of a single GeoJSON object.

3. GeoJSON Object

A GeoJSON object represents a geometry, feature, or collection of features.

- A GeoJSON object is a JSON object.
- A GeoJSON object MUST have a member with the name "type". The value of the member MUST be one of the GeoJSON types.
- A GeoJSON object MAY have a "bbox" member, the value of which MUST be a bounding box array (see Section 5).
- A GeoJSON object MAY have any number of other members (see Section 6).

3.1. Geometry Object

A Geometry object represents points, curves, and surfaces in coordinate space.

- The value of a geometry object's "type" member MUST be one of the seven geometry types (see Section 1.4).
- A GeoJSON geometry object of any type other than "GeometryCollection" MUST have a member with the name "coordinates". The value of the coordinates member is always an array. The structure of the elements in this array is determined by the type of geometry. GeoJSON processors MAY interpret geometry objects with empty coordinates arrays as null objects.

3.1.1. Position

A position is the fundamental geometry construct. The "coordinates" member of a geometry object is composed of either:

- one position (in the case of a Point geometry),
- an array of positions (LineString or MultiPoint geometries),
- an array of arrays of positions (Polygons, MultiLineStrings),
- or a multidimensional array of positions (MultiPolygon).

A position is an array of numbers. There MUST be two or more elements. The first two elements are longitude and latitude, or easting and northing, precisely in that order and using decimal numbers. Altitude or elevation MAY be included as an optional third element.

Implementations SHOULD NOT extend positions beyond 3 elements. Parsers MAY ignore additional elements. Interpretation and meaning of additional elements is beyond the scope of this specification.

A line between two positions is a straight Cartesian line, the shortest line between those two points in the Coordinate Reference System (see #Section 4).

In other words, every point on a line that does not cross the antimeridian between a point (lon0, lat0) and (lon1, lat1) can be calculated as

F(lon, lat) = (lon0 + (lon1 - lon0) * t, lat0 + (lat1 - lat0) * t)

with t a real number greater or equal to 0 and smaller or equal to 1.

Note that - for example in the WGS84 datum, the default Coordinate Reference System - this line may marketly differ from the geodesic path along the curved surface of the reference ellipsoid.

The same applies to the optional height element with the provisio that the direction of the height is as specified in the Coordinate Reference System.

Note that, again, using the default WGS84 datum as a starting point, this does not mean that a surface with equal height follows, for example, the curvature of a body of water. Nor is a surface of equal height perpendicular to a plum line.

Examples of positions and geometries are provided in "Appendix A. Geometry Examples".

3.1.2. Point

For type "Point", the "coordinates" member MUST be a single position.

3.1.3. MultiPoint

For type "MultiPoint", the "coordinates" member MUST be an array of positions.

3.1.4. LineString

For type "LineString", the "coordinates" member MUST be an array of two or more positions.

3.1.5. MultiLineString

For type "MultiLineString", the "coordinates" member MUST be an array of LineString coordinate arrays.

3.1.6. Polygon

To specify a constraint specific to polygons, it is useful to introduce the concept of a linear ring:

- A linear ring is a closed LineString with 4 or more positions.
- The first and last positions are equivalent, they MUST contain identical values; their representation SHOULD also be identical.
- A linear ring is the boundary of a surface or the boundary of a hole in a surface.
- A linear ring SHOULD follow right-hand rule with respect to the area it bounds (i.e., exterior rings are counter-clockwise, holes are clockwise)

Though a linear ring is not explicitly represented as a GeoJSON geometry type, it leads to a canonical formulation of the Polygon geometry type definition as follows:

- For type "Polygon", the "coordinates" member MUST be an array of linear ring coordinate arrays.
- For Polygons with more than one of these rings, the first MUST be the exterior ring and any others MUST be interior rings. The exterior ring bounds the surface and the interiors rings (if present) bound holes within the surface.

3.1.7. MultiPolygon

For type "MultiPolygon", the "coordinates" member MUST be an array of Polygon coordinate arrays.

3.1.8. Geometry Collection

A GeoJSON object with type "GeometryCollection" is a geometry object. A geometry collection MUST have a member with the name "geometries". The value of "geometries" is an array. Each element of this array is a GeoJSON geometry object. It is possible for this array to be empty.

Unlike the other geometry types described above, a geometry collection can be a heterogeneous composition of smaller geometry objects. For example, a geometry object in the shape of a lowercase roman "i" can be composed of one point and one line string.

3.2. Feature Object

A Feature object represents a spatially-bounded thing.

- A feature object MUST have a "type" member with the value "Feature".
- A feature object MUST have a member with the name "geometry". The value of the geometry member SHALL be either a geometry object as defined above or, in the the case that the feature is unlocated, a JSON null value.
- A feature object MUST have a member with the name "properties". The value of the properties member is an object (any JSON object or a JSON null value).
- If a feature has a commonly used identifier, that identifier SHOULD be included as a member of the feature object with the name "id" and the value of this member is either a JSON string or number.

3.3. Feature Collection Object

A GeoJSON object with the type "FeatureCollection" is a feature collection object. A feature collection object MUST have a member with the name "features". The value of "features" is a JSON array. Each element of the array is a feature object as defined above. It is possible for this array to be empty.

4. Coordinate Reference System

The default reference system for all GeoJSON coordinates SHALL be a geographic coordinate reference system, using the [WGS84] datum, and with longitude and latitude units of decimal degrees. This coordinate reference system is equivalent to the OGC's "http://

www.opengis.net/def/crs/OGC/1.3/CRS84" [OGCURL]. To maximize interoperability, GeoJSON data SHOULD use this default coordinate reference system. An OPTIONAL third position element SHALL be the height in meters above the WGS 84 reference ellipsoid. In the absence of elevation values, applications sensitive to height or depth SHOULD interpret positions as being at local ground or sea level.

Other coordinate reference systems, including ones described by CRS objects of the kind defined in [GJ2008] are NOT RECOMMENDED. GeoJSON processing software SHALL NOT be expected to have access to coordinate reference systems databases. Applications requiring a CRS other than the default MUST assume all responsibility for CRS identification, coordinate accuracy, and interpretation of missing elevation values. Furthermore, GeoJSON coordinates MUST NOT under any circumstances use latitude, longitude order.

5. Bounding Box

A GeoJSON object MAY have a member named "bbox" to include information on the coordinate range for its geometries, features, or feature collections. The value of the bbox member MUST be an array of length 2*n where n is the number of dimensions represented in the contained geometries, with all axes of the most south-westerly point followed by all axes of the more north-easterly point. The axes order of a bbox follows the axes order of geometries.

In the default GeoJSON CRS (see Section 4), the "bbox" values define shapes with edges that follow lines of constant longitude, latitude, and elevation.

Example of a 2D bbox member on a feature:

```
{
    "type": "Feature",
    "bbox": [-10.0, -10.0, 10.0, 10.0],
    "geometry": {
        "type": "Polygon",
        "coordinates": [
            [
                [-10.0, -10.0],
                [10.0, -10.0],
                [10.0, 10.0],
                [-10.0, -10.0]
            ]
        ]
    }
    //...
}
```

Example of a 2D bbox member on a feature collection:

```
{
    "type": "FeatureCollection",
    "bbox": [100.0, 0.0, 105.0, 1.0],
    "features": [
    //...
    ]
}
```

Example of a 3D bbox member with a depth of 100 meters:

```
{
    "type": "FeatureCollection",
    "bbox": [100.0, 0.0, -100.0, 105.0, 1.0, 0.0],
    "features": [
    //...
    ]
}
```

5.1. The connecting lines

The 4 lines of the bounding box are defined fully within the coordinate reference system; i.e. every point on the northernmost line can be expressed as

(lon, lat) = (%minlon% + (%maxlon% - %minlon%) * t, %maxlat%)

with 0 <= t <= 1.

5.2. The Antimeridian

Consider a set of point features within the Fiji archipelago, straddling the antimeridian between 16 degrees S and 20 degrees S. The southwest corner of the box containing these features is at 20 degrees S and 177 degrees E, the northwest corner is at 16 degrees S and 178 degrees W. In the default GeoJSON CRS (see Section 4) the antimeridian-spanning GeoJSON bounding box for this feature collection is

"bbox": [177.0, -20.0, -178.0, -16.0]

and covers 5 degrees of longitude.

The complementary bounding box for the same latitude band, not crossing the antimeridian, is

"bbox": [-178.0, -20.0, 177.0, -16.0]

and covers 355 degrees of longitude.

The latitude of the northeast corner is always greater than the latitude of the southwest corner, but bounding boxes that cross the antimeridian have a northeast corner longitude that is less than the longitude of the southwest corner.

5.3. The Poles

In the default GeoJSON CRS (see Section 4), a bounding box that contains the North Pole extends from a southwest corner of %minlat% degrees N, 180 degrees W to a northeast corner of 90 degrees N, 180 degrees E. Viewed on a globe, this bounding box approximates a spherical cap.

"bbox": [-180.0, %minlat%, 180.0, 90.0]

A bounding box that contains the South Pole extends from a southwest corner of 90 degrees S, 180 degrees W to a northeast corner of %maxlat% degrees S, 180 degrees E.

"bbox": [-180.0, -90.0, 180.0, %maxlat%]

A bounding box that just touches the North Pole and forms a slice of an approximate spherical cap when viewed on a globe has as its northeast corner coordinates the easternmost longitude value and 90 degrees N.

"bbox": [%westlon%, %minlat%, %eastlon%, 90.0]

A bounding box that just touches the South Pole and forms a slice of an approximate spherical cap when viewed on a globe has as its southwest corner coordinates the westernmost longitude value and 90 degrees S.

"bbox": [%westlon%, -90.0, %eastlon%, %maxlat%]

Implementers MUST NOT use latitude values greater than 90 or less than -90 when using the default GeoJSON coordinate reference system to imply an extent that is not a spherical cap.

6. Extending GeoJSON

6.1. Foreign members

Members not described in this specification ("foreign members") MAY be used in a GeoJSON document. Note that support for foreign members can vary across implementations and no normative processing model for foreign members is defined. Accordingly, implementations that rely too heavily on the use of foreign members might experience reduced interoperability with other implementations.

For example, in the (abridged) feature object shown below

```
{
    "type": "Feature",
    "id": "f1",
    "geometry": {...},
    "properties": {...},
    "title": "Example Feature",
}
```

the name/value pair of "title": "Example Feature" is a foreign member. When the value of a foreign member is an object, all the descendant members of that object are themselves foreign members.

GeoJSON semantics do not apply to foreign members and their descendants, regardless of their names and values. For example, in the (abridged) feature object below

```
{
    "type": "Feature",
    "id": "f2",
    "geometry": {...},
    "properties": {...},
    "centerline": {
        "type": "LineString",
        "coordinates": [
            [-170, 10],
            [170, 11]
        ]
    },
}
```

the "centerline" member is not a GeoJSON geometry object.

6.2. GeoJSON types are not extensible

Implementations MUST NOT extend the fixed set of GeoJSON types: FeatureCollection, Feature, Point, LineString, MultiPoint, Polygon, MultiLineString, MultiPolygon, and GeometryCollection.

6.3. Semantics of GeoJSON members and types are not changeable

Implementations MUST NOT change the the semantics of GeoJSON members and types.

The GeoJSON "coordinates" and "geometries" members define Geometry objects. FeatureCollection and Feature objects, respectively, MUST NOT contain a "coordinates" or "geometries" member.

The GeoJSON "geometry" and "properties" members define a Feature object. FeatureCollection and Geometry objects, respectively, MUST NOT contain a "geometry" or "properties" member.

The GeoJSON "features" member defines a FeatureCollection object. Feature and Geometry objects, respectively, MUST NOT contain a "features" member.

7. Versioning

The GeoJSON format can be extended as defined here, but no explicit versioning scheme is defined. A specification that alters the semantics of GeoJSON members or otherwise modifies the format does not create a new version of this format; instead, it defines an entirely new format that MUST NOT be identified as GeoJSON.

8. Mapping 'geo' URIs

'geo' URIs [RFC5870] identify geographic locations and precise (not uncertain) locations can be mapped to GeoJSON geometry objects.

For this section, as in [RFC5870], "%lat%", "%lon%", "%alt%", and "%unc%" are placeholders for 'geo' URI latitude, longitude, altitude, and uncertainty values, respectively.

A 'geo' URI with two coordinates and an uncertainty ('u') parameter that is absent or zero, and a GeoJSON Point geometry may be mapped to each other. A GeoJSON point is always converted to a 'geo' URI that has no uncertainty parameter.

'geo' URI:

geo:%lat%,%lon%

GeoJSON:

{"type": "Point", "coordinates": [%lon%, %lat%]}

The mapping between 'geo' URIs and GeoJSON points that specify elevation is shown below.

'geo' URI:

geo:%lat%,%lon%,%alt%

GeoJSON:

{"type": "Point", "coordinates": [%lon%, %lat%, %alt%]}

GeoJSON has no concept of uncertainty; imprecise or uncertain 'geo' URIs thus can not be mapped to GeoJSON geometries.

9. Security Considerations

GeoJSON shares security issues common to all JSON content types. See [RFC7159] Section 12 for additional information. GeoJSON does not provide executable content.

As with other geographic data formats, e.g., [KMLv2.2], providing details about the locations of sensitive persons, animals, habitats, and facilities can expose them to unauthorized tracking or injury. GeoJSON does not provide privacy or integrity services; if sensitive data requires privacy or integrity protection the service must be provided externally.

10. Interoperability Considerations

10.1. I-JSON

GeoJSON texts SHOULD follow the constraints of I-JSON [RFC7493] for maximum interoperability.

10.2. Coordinate Precision

The size of a GeoJSON text in bytes is a major interoperability consideration and precision of coordinate values has a large impact on the size of texts. A GeoJSON text containing many detailed polygons can be inflated almost by a factor of two by increasing coordinate precision from 6 to 15 decimal places. For geographic coordinates with units of degrees, 6 decimal places (a default common in, e.g., sprintf) amounts to about 10 centimeters, a precision well within that of current GPS systems. Implementations should consider the cost to using a greater precision than necessary.

Furthermore the default WGS84 datum uses a relatively coarse geoid; with the WGS84 height varying by up to 5m (but generally between 2 and 3 meter) higher or lower relative to a surface parallel to Earth's mean sea level.

10.3. Coordinate Order

There are conflicting precedents among geographic data formats over whether latitude or longitude come first in a pair of numbers. Longitude comes first in GeoJSON coordinates as it does in [KMLv2.2].

Some commonly-used CRS definitions specify coordinate ordering that is not longitude then latitude (for a geographic CRS) or easting then northing (for a projected CRS). The CRS historically known as "EPSG:4326" and more accurately identified by "http://www.opengis.net/def/crs/EPSG/0/4326" is a prime example. Using such a CRS is NOT RECOMMENDED due to the potential disruption of interoperability. When such a CRS is encountered in GeoJSON, the document should be processed with caution. Heuristics may be necessary to interpret the coordinates properly; they may not be in the required longitude, latitude order.

10.4. Antimeridian cutting

In representing features that cross the antimeridian, interoperability is improved by cutting geometries so that no single part crosses the antimeridian. For example, a line extending from 45 degrees N, 170 degrees E across the antimeridian to 45 degrees N, 170 degrees W should be cut in two and represented as a MultiLineString.

```
{"type": "MultiLineString", "coordinates": [
  [[170.0, 45.0], [180.0, 45.0]],
  [[-180.0, 45.0], [-170.0, 45.0]]]}
```

A rectangle extending from 40 degrees N, 170 degrees E across the antimeridian to 50 degrees N, 170 degrees W should be cut in two and represented as a MultiPolygon.

```
{"type": "MultiPolygon", "coordinates": [
  [[[180.0, 40.0], [180.0, 50.0], [170.0, 50.0], [170.0, 40.0],
    [180.0, 40.0]]],
  [[[-170.0, 40.0], [-170.0, 50.0], [-180.0, 50.0], [-180.0, 40.0],
    [-170.0, 40.0]]]]}
```

10.5. Geometry Collections

Geometry collections have a different syntax from single type geometry objects (points, line strings, and polygons) and homogeneously typed multipart geometry objects (multipoints, multiline strings, and multipolygons) but have no different semantics. Although a geometry collection object has no "coordinates" member, it does have coordinates: the coordinates of all its parts belong to the collection. The "geometries" member of a geometry collection describes the parts of this composition. Implementations SHOULD NOT apply any additional semantics to the "geometries" array.

To maximize interoperability implementations SHOULD avoid nested geometry collections. Furthermore, geometry collections composed of a single part or a number of parts of a single SHOULD be avoided when that single part or a single object of multi-part type (MultiPoint, MultiLineString, or MultiPolygon) could be used instead.

11. IANA Considerations

The media type for GeoJSON text is "application/geo+json" and is registered in the "Media Types" registry described in [RFC6838]. The entry for "application/vnd.geo+json" in the same registry should have its status changed to be Obsolete with a pointer to the media type "application/geo+json" and a reference added to this RFC.

Type name: application

Subtype name: geo+json

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: binary

Security considerations: See section 5 above

Interoperability considerations: See section 6 above

Published specification: [[This document]]

Applications that use this media type: various

Additional information:

Magic number(s): n/a

File extension(s): .json, .geojson

Macintosh file type code: n/a

Object Identifiers: n/a

Windows clipboard name: GeoJSON

Macintosh uniform type identifier: public.geojson conforms to public.json

Person to contact for further information: Sean Gillies (sean.gillies@gmail.com)

Intended usage: COMMON

Restrictions on usage: none

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <http://www.rfc-editor.org/info/rfc7493>.

[WGS84] National Imagery and Mapping Agency, "Department of Defense World Geodetic System 1984, Third Edition", 1984.

12.2. Informative References

[GJ2008] Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., and C. Schmidt, "The GeoJSON Format Specification", June 2008.

[KMLv2.2] Wilson, T., "OGC KML", OGC 07-147r2, April 2008.

[OGCURL] Cox, S., "OGC-NA Name type specification - definitions: Part 1 - basic name", OGC 09-048r3, March 2010.

[RFC7464] Williams, N., "JavaScript Object Notation (JSON) Text Sequences", RFC 7464, DOI 10.17487/RFC7464, February 2015, <http://www.rfc-editor.org/info/rfc7464>.

[SFSQL] OpenGIS Consortium, Inc., "OpenGIS Simple Features Specification For SQL Revision 1.1", OGC 99-049, May 1999.

[WFSv1] Vretanos, P., "Web Feature Service Implementation Specification", OGC 02-058, May 2002.

Appendix A. Geometry Examples

Each of the examples below represents a valid and complete GeoJSON object.

A.1. Points

Point coordinates are in x, y order (easting, northing for projected coordinates, longitude, latitude for geographic coordinates):

```
{
    "type": "Point",
    "coordinates": [100.0, 0.0]
}
```

A.2. LineStrings

Coordinates of LineString are an array of positions (see "2.1.1. Position"):

```
{
    "type": "LineString",
    "coordinates": [
        [100.0, 0.0],
        [101.0, 1.0]
    ]
}
```

A.3. Polygons

Coordinates of a Polygon are an array of LinearRing (cf. "2.1.6 Polygon") coordinate arrays. The first element in the array represents the exterior ring. Any subsequent elements represent interior rings (or holes).

No holes:

```
{
    "type": "Polygon",
    "coordinates": [
        [
            [100.0, 0.0],
            [101.0, 0.0],
            [101.0, 1.0],
            [100.0, 1.0],
            [100.0, 0.0]
        ]
    ]
}
```

With holes:

```
    {
        "type": "Polygon",
        "coordinates": [
            [
                [100.0, 0.0],
                [101.0, 0.0],
                [101.0, 1.0],
                [100.0, 1.0],
                [100.0, 0.0]
            ],
            [
                [100.8, 0.8],
                [100.8, 0.2],
                [100.2, 0.2],
                [100.2, 0.8],
                [100.8, 0.8]
            ]
        ]
    }
```

With hole crossing dateline:

```
    {
        "type": "Polygon",
        "coordinates": [
            [
                [-170.0, 10.0],
                [170.0, 10.0],
                [170.0, -10.0],
                [-170.0, -10.0],
                [-170.0, 10.0]
            ],
            [
                [175.0, 5.0],
                [-175.0, 5.0],
                [-175.0, -5.0],
                [175.0, -5.0],
                [175.0, 5.0]
            ]
        ]
    }
```

A.4. MultiPoints

Coordinates of a MultiPoint are an array of positions::

```
{
    "type": "MultiPoint",
    "coordinates": [
        [100.0, 0.0],
        [101.0, 1.0]
    ]
}
```

A.5. MultiLineStrings

Coordinates of a MultiLineString are an array of LineString coordinate arrays:

```
{
    "type": "MultiLineString",
    "coordinates": [
        [
            [100.0, 0.0],
            [101.0, 1.0]
        ],
        [
            [102.0, 2.0],
            [103.0, 3.0]
        ]
    ]
}
```

A.6. MultiPolygons

Coordinates of a MultiPolygon are an array of Polygon coordinate arrays:

```
    {
        "type": "MultiPolygon",
        "coordinates": [
            [
                [
                    [102.0, 2.0],
                    [103.0, 2.0],
                    [103.0, 3.0],
                    [102.0, 3.0],
                    [102.0, 2.0]
                ]
            ],
            [
                [
                    [100.0, 0.0],
                    [101.0, 0.0],
                    [101.0, 1.0],
                    [100.0, 1.0],
                    [100.0, 0.0]
                ],
                [
                    [100.2, 0.2],
                    [100.8, 0.2],
                    [100.8, 0.8],
                    [100.2, 0.8],
                    [100.2, 0.2]
                ]
            ]
        ]
    }
```

A.7. GeometryCollections

Each element in the geometries array of a GeometryCollection is one of the geometry objects described above:

```
{
    "type": "GeometryCollection",
    "geometries": [{
        "type": "Point",
        "coordinates": [100.0, 0.0]
    }, {
        "type": "LineString",
        "coordinates": [
            [101.0, 0.0],
            [102.0, 1.0]
        ]
    }]
}
```

Appendix B. Changes from pre-IETF specification

This appendix briefly summarizes non-editorial changes from the 2008 specification [GJ2008].

B.1. Normative changes

- Coordinate reference systems other than the default are NOT RECOMMENDED (see Section 4).
- In the absence of elevation values, applications sensitive to height or depth SHOULD interpret positions as being at local ground or sea level (see Section 4).
- Implementations SHOULD NOT extend position arrays beyond 3 elements (see Section 3.1.1).
- A line between two positions is a straight Cartesian line (see Section 3.1.1).
- The values of a "bbox" array are "[%west%, %south%, %east%, %north%]", not [%minx%, %miny%, %maxx%, %maxy%]` (see Section 5).
- Extensions MAY be used, but MUST NOT change the the semantics of GeoJSON members and types (see Section 6).
- GeoJSON objects MUST NOT contain the defining members of other types (see Section 6.3).
- The media type for GeoJSON is application/geo+json.

B.2. Informative changes

- The definition of a GeoJSON text has been added.
- Rules for mapping 'geo' URIs have been added.
- A recommendation of the I-JSON [RFC7493] constraints has been added.
- Implementers are cautioned about the effect of excessive coordinate precision on interoperability.
- Right-hand rule orientation of polygon rings (counter-clockwise external rings, clockwise internal rings) is recommended to improve interoperability.
- Interoperability concerns of geometry collections are noted. These objects should be used sparingly (see Section 10.5).

Appendix C. GeoJSON Text Sequences

All GeoJSON objects defined in this specification - FeatureCollection, Feature, and Geometry - consist of exactly one JSON object. However, there may be circumstances in which applications need to represent sets or sequences of these objects (over and above the grouping of Feature objects in a FeatureCollection), e.g. in order to efficiently "stream" large numbers of Feature objects. The definition of such sets or sequences is outside the scope of this specification.

If such a representation is needed, a new media type is required that has the ability to represent these sets or sequences. When defining such a media type, it may be useful to base it on "JSON Text Sequences" [RFC7464], leaving the foundations of how to represent multiple JSON objects to that specification, and only defining how it applies to GeoJSON objects.

Appendix D. Contributors

The GeoJSON format is the product of discussion on the GeoJSON mailing list, http://lists.geojson.org/listinfo.cgi/geojson-geojson.org, before October 2015 and the IETF's GeoJSON WG after October 2015.

Comments are solicited and should be addressed to the GeoJSON mailing list at geojson@ietf.org or to the GeoJSON issue tracker at https://github.com/geojson/draft-geojson/issues.

Authors' Addresses

H. Butler
Hobu Inc.

Email: howard@hobu.co

M. Daly
Cadcorp

Email: martin.daly@cadcorp.com

A. Doyle
MIT

Email: adoyle@intl-interfaces.com

S. Gillies
Mapbox Inc.

Email: sean.gillies@gmail.com
URI: http://sgillies.net

T. Schaub
Planet Labs

Email: tim.schaub@gmail.com

S. Hagen
Rheinaustr. 62
Bonn 53225
DE

Email: stefan@hagen.link
URI: http://stefan-hagen.website/