Geospatial File Formats

OpenSHA stores geospatial data in GeoJSON, which is specified in RFC 7946. OpenSHA code for (de)serializing GeoJSON can be found in the *org.opensha.commons.geo.json* package.

Important note on depths/elevations: *OpenSHA stores depth data in km, positive down. For example, a value of* `3.0` *in the third column of a coordinate array indicates that the given point is 3 km below the local surface. This differs from the GeoJSON specification, but is in-line with the USGS Event Web Service and general treatment of 3D location data in OpenSHA and other PSHA codes. You can load a file in code that uses a different depth standard with the* `buildGson(DepthSerializationType)` *method of FeatureCollection.*

Table of Contents

Fault Data

(return to top)

Fault data are stored as GeoJSON `Feature` objects, and a collection of faults (e.g., a fault model or fault subsection list) are stored in a `FeatureCollection`. See GeoJSONFaultSection for the OpenSHA implementation of this format.

Fault data requirements

(return to top)

At a minimum, a GeoJSON fault must contain the **3 following items**:

1. Fault Trace

(return to top)

The fault trace must be present in the `geometry` object in the form of a `LineString` or `MultiLineString`. If a `MultiLineString` is encountered and it contains a single `LineString` (GIS softwares may output single lines in this format), it is supported. If a `MultiLineString` is encountered and it contains two `LineString`s, then the second trace is treated as a lower trace; that lower trace must be below the upper trace (it must explicitly specify depths in the `coordinates` array), and must be in the same general direction as the upper trace.

Example fault trace as a `LineString` with 2 points:

```
"geometry": {
  "type": "LineString",
  "coordinates": [
    [ -117.74953000000001, 35.74054 ],
    [ -117.76365068593667, 35.81037829696144 ]
  ]
}
```

Example upper and lower fault traces as a `MultiLineString`, each with explicitly stated depths in the `coordinates` array:

```
"geometry": {
  "type": "MultiLineString",
  "coordinates": [
    [
      [ -117.74953000000001, 35.74054, 0.0 ],
      [ -117.76365068593667, 35.81037829696144, 0.0 ]
    ],
    [
      [ -117.74953000000001, 35.74054, 10.0 ],
      [ -117.76365068593667, 35.81037829696144, 10.0 ]
    ]
  ]
}
```

2. Required properties

(return to top)

The following `properties` are required:

Name	JSON Type	Description
DipDeg	Number	

Name	JSON Type	Description
		Dip of the fault in decimal degrees, following the right hand rule. See the glossary for more information. Note: this can be omitted for dipping faults if a lower fault trace is supplied.
LowDepth	Number	Lower depth of the fault in kilometers. See simple fault for more information. Note: this can be omitted for dipping faults if a lower fault trace is supplied.
Rake	Number	Rake of the fault in decimal degrees, see the glossary for more information.
UpDepth	Number	Upper depth of the fault in kilometers, not including any aseismicity. See simple fault for more information. Note: this can be omitted for dipping faults if a lower fault trace is supplied.

3. Unique ID

(return to top)

A unique non-negative integer ID is required for each fault section. This can be specified either as the `id` field of the `Feature` itself (must be an integer), or via the optional `FaultID` property. If both exist, the `id` field of the feature takes precedence.

Fault data optional extensions

(return to top)

The following optional properties will be parsed by OpenSHA (other properties may be present and will be ignored):

Name	JSON Type	Description	Default Value
`AseismicSlipFactor`	Number	Fraction (value in the range `[0,1)`) of the fault area that is aseismic, typically applied by increasing the upper depth of the fault such that the area is reduced by this fraction.	`0.0`
`Connector`	Boolean	Boolean indicating that this fault is a Connector (currently unused).	*(none)*
`CouplingCoeff`	Number	Fraction (value in the range `[0,1]`) of the slip rate of this fault that is released seismically.	`1.0`
`DipDir`	Number	Dip direction of this fault, see the glossary for more information.	Average trace strike direction + 90 degrees
`DateLastEvent`	Number	Date of the last event that ruptured this fault, used in time-dependent forecasts, expressed in epoch milliseconds.	*(none)*
`FaultID`	Number	Integer ID of this fault. Must supply either this or the `Feature`'s `id` field.	*(none)*
`FaultName`	String	Name of this fault.	*(none)*
`ParentID`	Number	Integer ID of the parent to this fault. This is typically used when subdividing a fault into subsections, and will point to the ID of the original fault section.	*(none)*
`ParentName`	String	Name of the parent to this fault. This is typically used when	*(none)*

Name	JSON Type	Description	Default Value
		subdividing a fault into subsections, and will give the name of the original fault section.	
`PrimState`	String	2 letter abbreviation of the primarily associated US state for this fault, if it exists.	*(none)*
`SecState`	String	2 letter abbreviation of the secondary associated US state for this fault, if it exists.	*(none)*
`SlipLastEvent`	Number	Slip in meters of the last event that ruptured this fault.	*(none)*
`SlipRate`	Number	Average long-term on-plane slip rate of this fault in mm/yr.	*(none)*
`SlipRateStdDev`	Number	Standard deviation of the average long-term slip rate of this fault in mm/yr.	*(none)*

You can optionally supply a polygon geometry that this fault represents. In this case, the `geometry` object must be a `GeometryCollection` that contains both a fault trace (as either a `LineString` or `MultiLineString`) and a polygon as either a `Polygon` or `MultiPolygon`. For example:

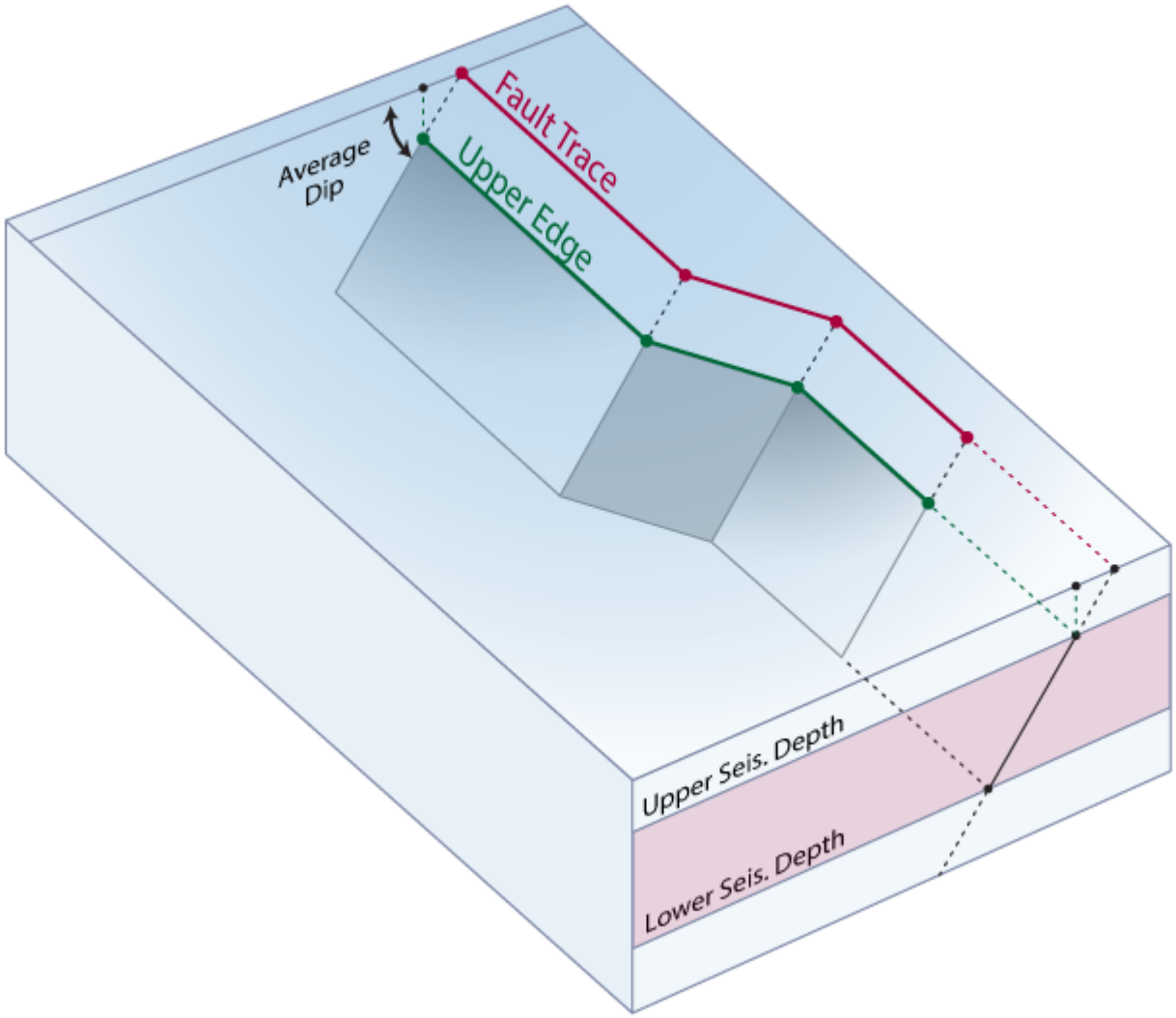
```
"geometry": {
  "type": "GeometryCollection",
  "geometries": [
    {
      "type": "LineString",
      "coordinates": [
        [ -117.76365068593667, 35.81037829696144 ],
        [ -117.76492000000002, 35.81665 ],
        [ -117.7758769984411, 35.880450900949334 ]
      ]
```

```
    },
    {
      "type": "Polygon",
      "coordinates": [
        [
          [ -117.73341200000002, 36.16374 ],
          [ -117.75440599999999, 36.158123 ],
          [ -117.76325900000002, 36.159714 ],
          [ -117.77182599999999, 36.112068 ],
          [ -117.779052, 36.092946 ],
          [ -117.73341200000002, 36.16374 ]
        ]
      ]
    }
  ]
}
```


Note on fault traces for buried, dipping faults

(return to top)

The notion of a *fault trace* is complicated for buried dipping faults (i.e., those with dip < 90 and upper seismogenic depth > 0). In OpenSHA, we typically assume this simple fault geometry where a rupture surface is defined by the down-dip projection of the *fault trace* (shown in red at the earth's surface in the schematic below):



Simple fault example

Modelers may wish, instead, to specify the *upper edge* of the (buried) surface (shown in green in the schematic above) rather than the up-dip extension at the earth's surface (shown in red). To accommodate this, we adopt the following convention when reading fault section data from GeoJSON:

If trace locations are specified without depths in the GeoJSON file, e.g.,:

```
"coordinates": [ [ -120.7585, 36.79945 ], [ -120.70175, 36.71373 ], [ -120.64514, 36.62798 ] ]
```

...then the trace is assumed to be an *upper edge* and the depths are set to the upper seismogenic depth. So, for example, if the "UpDepth" property (upper seismogenic depth) is set to 5 km, then the above supplied trace would be interpreted as:

```
"coordinates": [ [ -120.7585, 36.79945, 5.0 ], [ -120.70175, 36.71373, 5.0 ], [ -120.64514, 36.62798,
        5.0 ] ]
```

If, instead, the locations are specified with depths, e.g.,:

```
"coordinates": [ [ -120.7585, 36.79945, 0.0 ], [ -120.70175, 36.71373, 0.0 ], [ -120.64514, 36.62798, 0.
        0 ] ]
```

Or another example, this time with non-zero depths:

```
"coordinates": [ [ -120.7585, 36.79945, 3.0 ], [ -120.70175, 36.71373, 3.0 ], [ -120.64514, 36.62798,
        3.0 ] ]
```

...then the trace will be placed at the supplied depth, and the surface will be projected down-dip from that trace. In this case, the supplied depth must at or above the upper seismogenic depth. GeoJSON files generated by OpenSHA codes will always have the depth of fault traces explicitly stated for buried dipping faults, so this convention only applies to externally-generated files.

Example Fault Data GeoJSON

(return to top)

Here is an example `FeatureCollection` that contains a single fault, represented as a `Feature`:


```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "id": 0,
      "properties": {
        "FaultID": 0,
        "FaultName": "Airport Lake, Subsection 0",
        "DipDeg": 50.0,
        "Rake": -90.0,
        "LowDepth": 13.0,
        "UpDepth": 0.0,
        "DipDir": 89.4594,
        "AseismicSlipFactor": 0.1,
        "CouplingCoeff": 1.0,
        "SlipRate": 0.39,
        "ParentID": 861,
        "ParentName": "Airport Lake",
        "SlipRateStdDev": 0.0
      },
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [ -117.74953000000001, 35.74054 ],
          [ -117.76365068593667, 35.81037829696144 ]
        ]
      }
    }
  ]
}
```

Regions

(return to top)

OpenSHA Regions are stored as GeoJSON `Feature` elements that include a `Polygon` or `MultiPolygon`. `MultiPolygon`'s are supported to increase compatibility, but must consist of a single `Polygon`.

A Region can have a name, which is stored in the `id` field of the `Feature` as a JSON string.

Polygons should follow the GeoJSON specification, notably that they shall contain at least one linear ring (a closed path). The first path shall be the exterior ring (should be ordered counterclockwise according to the RFC, but we don't check) and subsequent rings are considered interiors (paths) and should be clockwise (though again, we don't check).

Region Example

(return to top)

Here is an example region that is a simple rectangle with minLat=34, maxLat=36, minLon=-120, and maxLon=-118:

```
{
  "type": "Feature",
  "id": "Simple region",
  "properties": {},
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [ -120.0, 34.0 ],
        [ -118.0, 34.0 ],
        [ -118.0, 36.0 ],
```

```
        [ -120.0, 36.0 ],
        [ -120.0, 34.0 ]
      ]
    ]
  }
}
```

Region Example With a Hole

(return to top)

Here is an example region that is a rectangle with minLat=34, maxLat=36, minLon=-120, and maxLon=-118, with an interior (hole) cut out with minLat=34.5, maxLat=35.5, minLon=-119.5, and maxLon=-118.5:

```
{
  "type": "Feature",
  "id": "Region with a hole",
  "properties": {},
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [ -120.0, 34.0 ],
        [ -118.0, 34.0 ],
        [ -118.0, 36.0 ],
        [ -120.0, 36.0 ],
        [ -120.0, 34.0 ]
      ],
      [
        [ -119.5, 34.5 ],
        [ -119.5, 35.5 ],
        [ -118.5, 35.5 ],
        [ -118.5, 34.5 ],
```

```
        [ -119.5, 34.5 ]
      ]
    ]
  }
}
```

Gridded Regions

(return to top)

OpenSHA Gridded Regions are stored as GeoJSON `Feature` elements that include a `Polygon` or `MultiPolygon` to specify the region boundary, and a `PointCollection` to specify grid nodes. Grid node locations must be evenly discretized in latitude in longitude, though the latitude and longitude spacing can be different. They can be irregular for non-rectangular regions.

Like a Region, a Gridded Region can have a name, which is stored in the `id` field of the `Feature` as a JSON string.

Gridded Region Optional Properties

(return to top)

The following are optional properties, used primarily for OpenSHA bookkeeping, and can be safely omitted so long as the Gridded Region does not contain any holes. If omitted, they will be inferred from the supplied grid nodes.

Name	JSON Type	Description
`Anchor`	Array of Number	Lon,Lat of the anchor (lower left) point of the grid
`LatNodes`	Array of Number	List of unique grid node latitudes in increasing order

Name	JSON Type	Description
`LatSpacing`	Number	Latitude grid spacing in decimal degrees
`LonNodes`	Array of Number	List of unique grid node longitudes in increasing order
`LonSpacing`	Number	Longitude grid spacing in decimal degrees

Gridded Region Example

(return to top)

Simple example:

```
{
  "type": "Feature",
  "id": "Example gridded region",
  "geometry": {
    "type": "GeometryCollection",
    "geometries": [
      {
        "type": "Polygon",
        "coordinates": [
          [
            [ -120.0, 34.0 ],
            [ -118.0, 34.0 ],
            [ -118.0, 36.0 ],
            [ -120.0, 36.0 ],
            [ -120.0, 34.0 ]
          ]
        ]
      },
      {
        "type": "MultiPoint",
```

```
        "coordinates": [
          [ -120.0, 34.0 ],
          [ -119.5, 34.0 ],
          [ -119.0, 34.0 ],
          [ -118.5, 34.0 ],
          [ -118.0, 34.0 ],
          [ -120.0, 34.5 ],
          [ -119.5, 34.5 ],
          [ -119.0, 34.5 ],
          [ -118.5, 34.5 ],
          [ -118.0, 34.5 ],
          [ -120.0, 35.0 ],
          [ -119.5, 35.0 ],
          [ -119.0, 35.0 ],
          [ -118.5, 35.0 ],
          [ -118.0, 35.0 ],
          [ -120.0, 35.5 ],
          [ -119.5, 35.5 ],
          [ -119.0, 35.5 ],
          [ -118.5, 35.5 ],
          [ -118.0, 35.5 ],
          [ -120.0, 36.0 ],
          [ -119.5, 36.0 ],
          [ -119.0, 36.0 ],
          [ -118.5, 36.0 ],
          [ -118.0, 36.0 ]
        ]
      }
    ]
  }
}
```

Example with full optional OpenSHA metadata properties:

```
{
  "type": "Feature",
  "id": "Example gridded region",
  "properties": {
    "LatNodes": [ 34.0, 34.5, 35.0, 35.5, 36.0 ],
    "LonNodes": [ -120.0, -119.5, -119.0, -118.5, -118.0 ],
    "LatSpacing": 0.5,
    "LonSpacing": 0.5,
    "Anchor": [ -120.0, 34.0 ]
  },
  "geometry": {
    "type": "GeometryCollection",
    "geometries": [
      {
        "type": "Polygon",
        "coordinates": [
          [
            [ -120.0, 34.0 ],
            [ -118.0, 34.0 ],
            [ -118.0, 36.0 ],
            [ -120.0, 36.0 ],
            [ -120.0, 34.0 ]
          ]
        ]
      },
      {
        "type": "MultiPoint",
        "coordinates": [
          [ -120.0, 34.0 ],
          [ -119.5, 34.0 ],
          [ -119.0, 34.0 ],
          [ -118.5, 34.0 ],
          [ -118.0, 34.0 ],
          [ -120.0, 34.5 ],
```

```
          [ -119.5, 34.5 ],
          [ -119.0, 34.5 ],
          [ -118.5, 34.5 ],
          [ -118.0, 34.5 ],
          [ -120.0, 35.0 ],
          [ -119.5, 35.0 ],
          [ -119.0, 35.0 ],
          [ -118.5, 35.0 ],
          [ -118.0, 35.0 ],
          [ -120.0, 35.5 ],
          [ -119.5, 35.5 ],
          [ -119.0, 35.5 ],
          [ -118.5, 35.5 ],
          [ -118.0, 35.5 ],
          [ -120.0, 36.0 ],
          [ -119.5, 36.0 ],
          [ -119.0, 36.0 ],
          [ -118.5, 36.0 ],
          [ -118.0, 36.0 ]
        ]
      }
    ]
  }
}
```