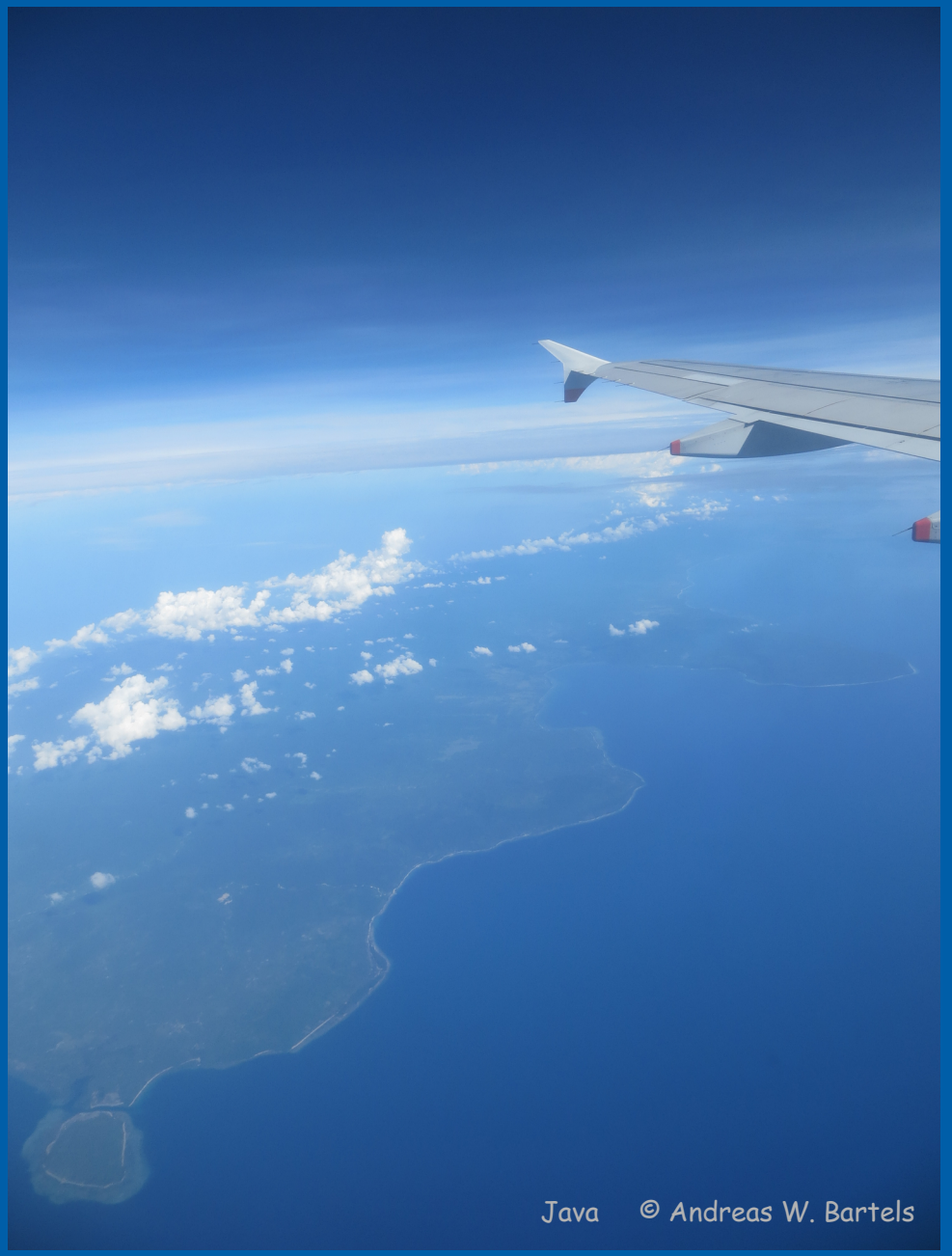
JSON Schema Driven Development in Java mit JSSD

Andreas W. Bartels

JUG Karlsruhe 16.11.2016

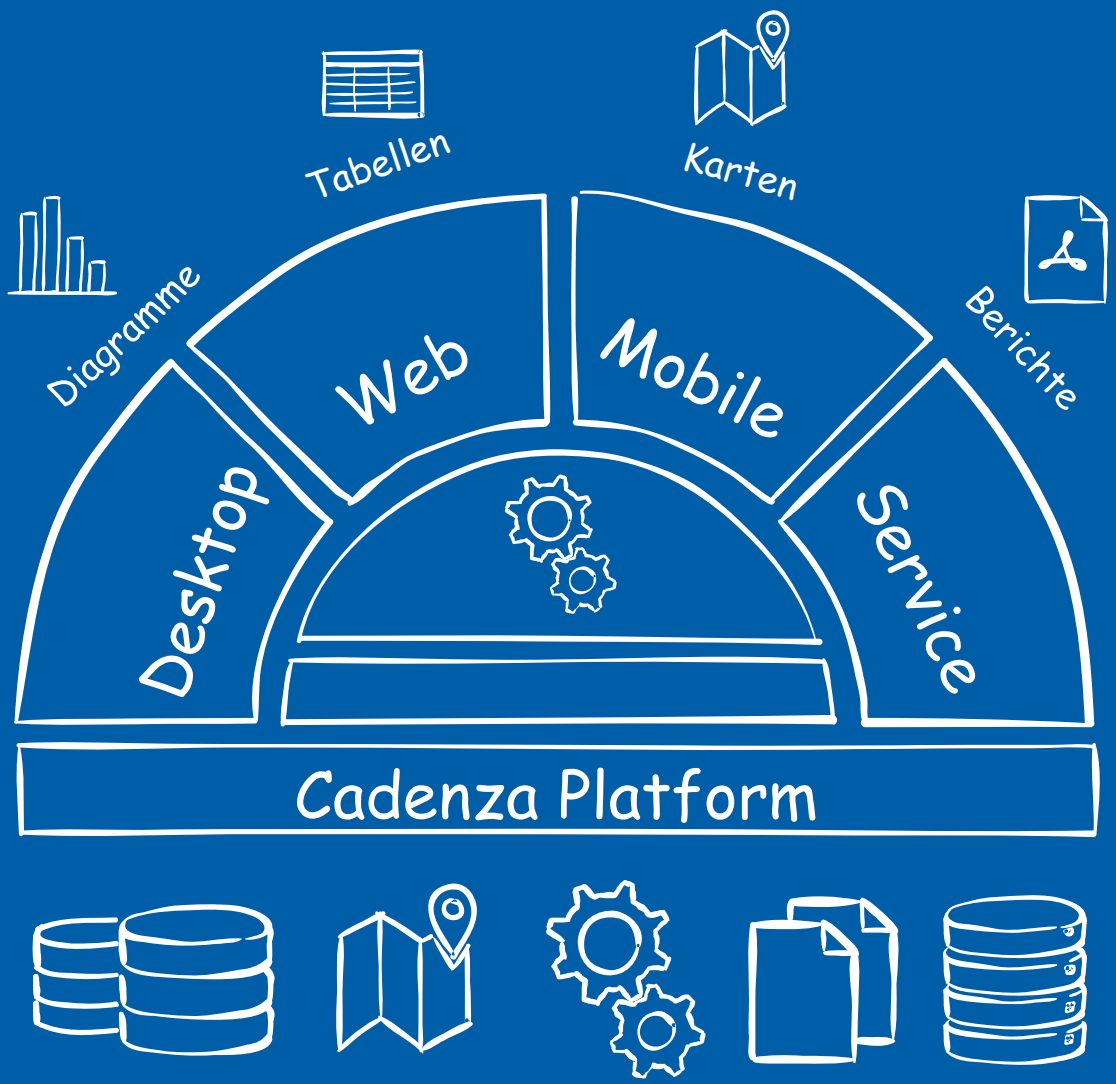


Andreas W. Bartels

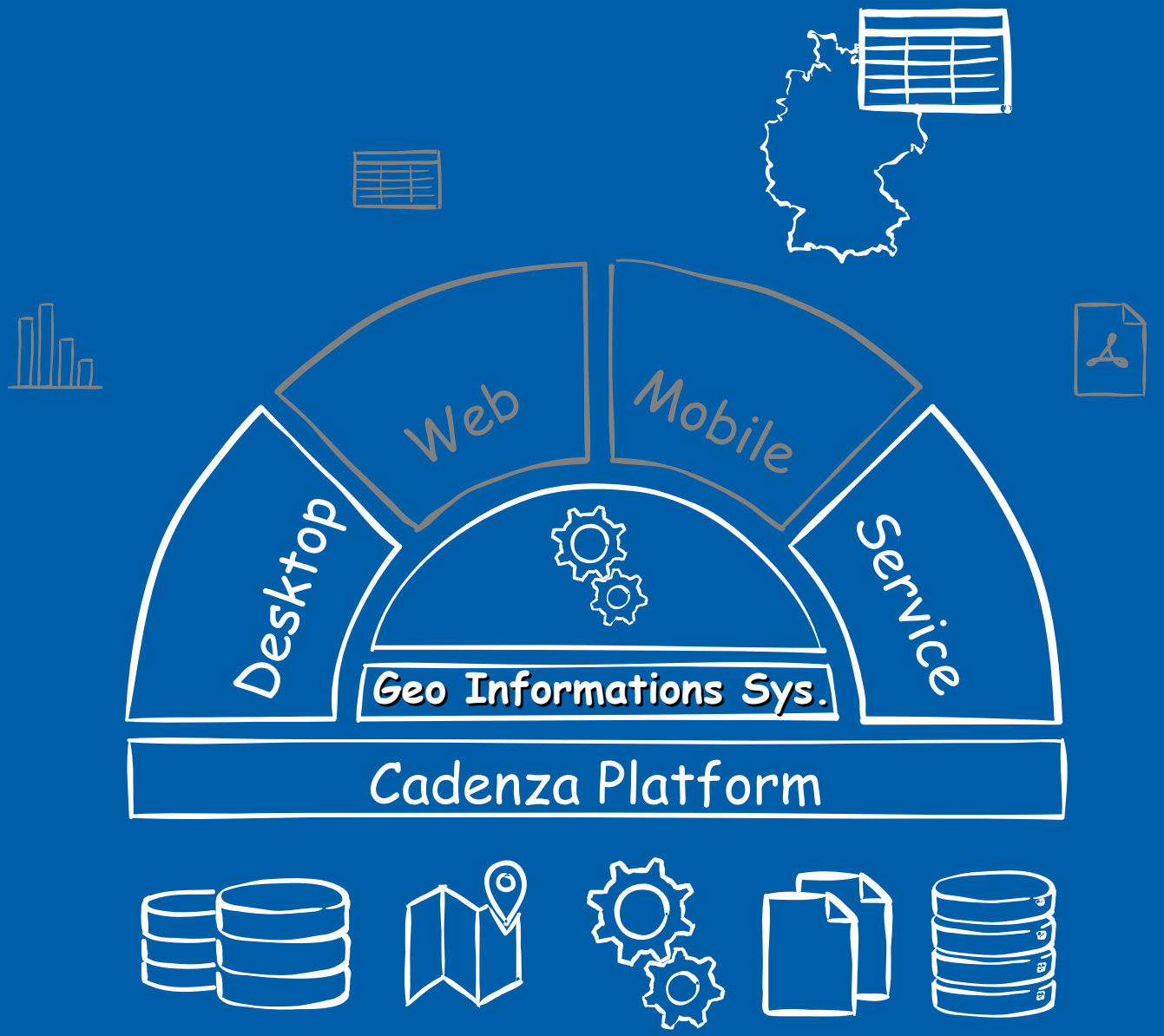
Disy Informationssysteme GmbH
Software Architekt

andreas.bartels@disy.net
https://github.com/AndreasWBartels

Disy


Tabellen
Karten
Diagramme
Berichte
Desktop
Web
Mobile
Service
Cadenza Platform

Ich bei Disy


Desktop
Web
Mobile
Service
Geo Informations Sys.
Cadenza Platform

JGISShell

47 lines (28 sloc) | 2.03 KB

Raw | Blame | History

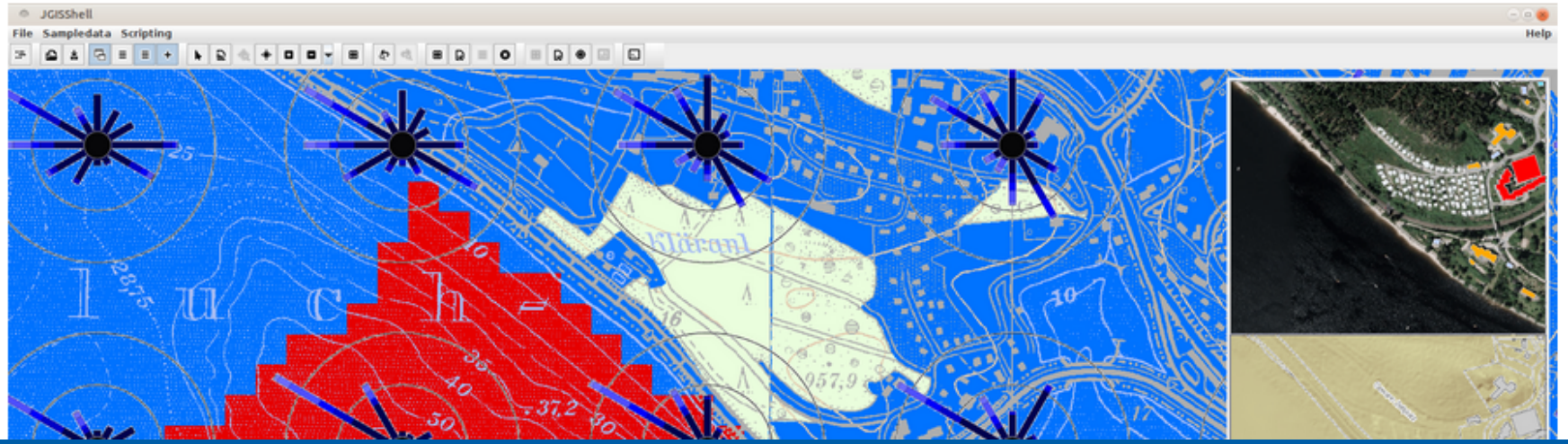
JGISShell

Java GIS Viewer and Layer Datastore Manager

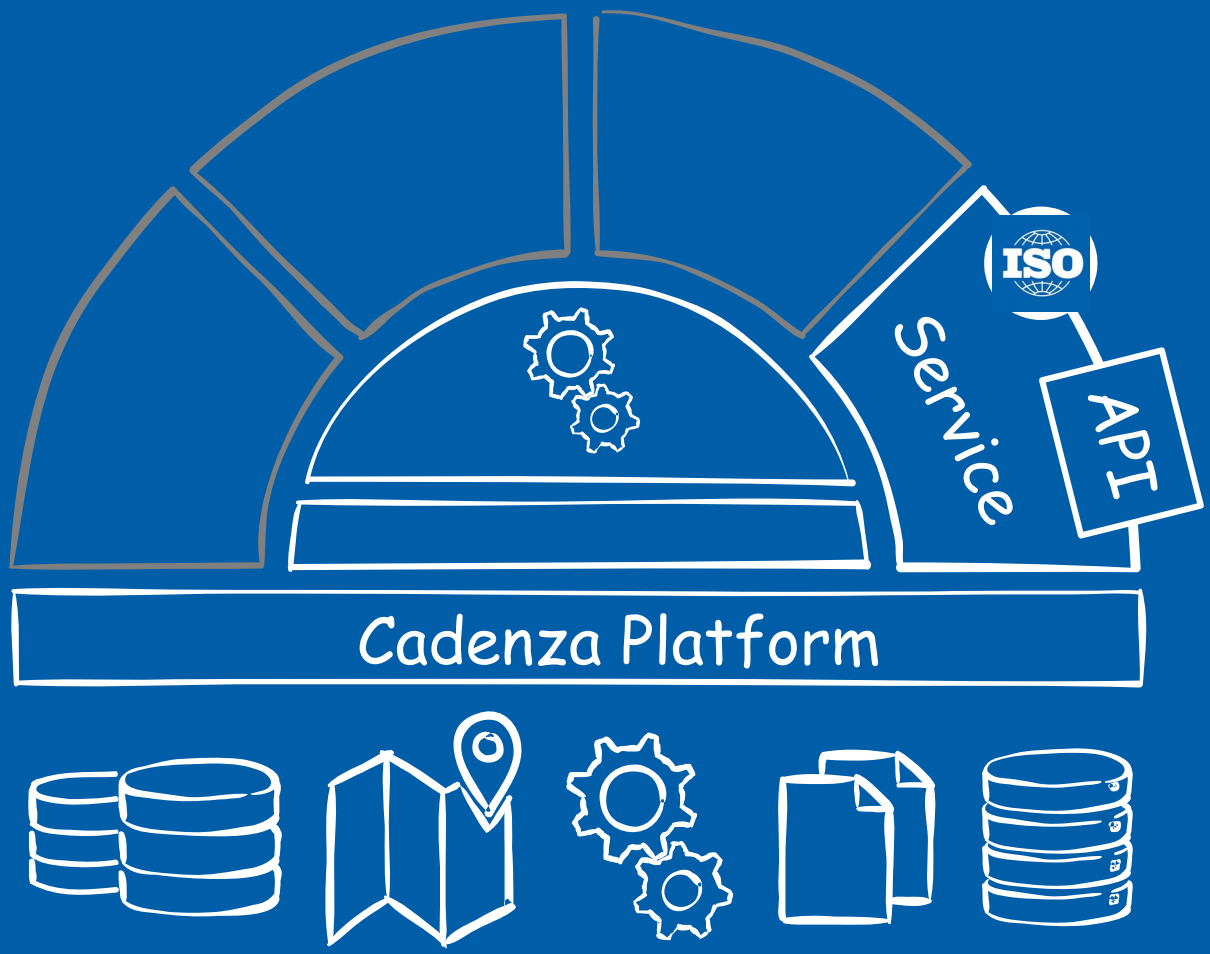
JGSShell is a Java based spatial data viewer and layer datastore manager. It consist of 4 parts, a viewer, a layer manager, a scripting API (groovy and Java) and a URL concept to access layer and datastores.

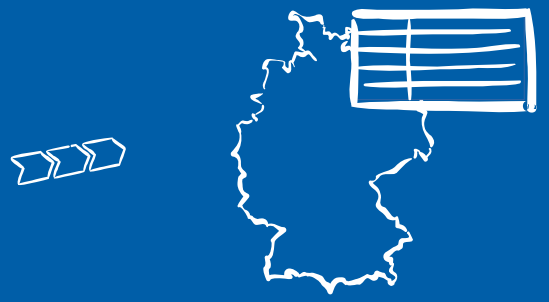
I started JGISShell 10 years ago as an private education and evaluation project for spatial data and technologies. Now it has riched a state that it could be used by other. Maybe for educaction, explore spatial data or manage spatial data.

Geodata Viewer



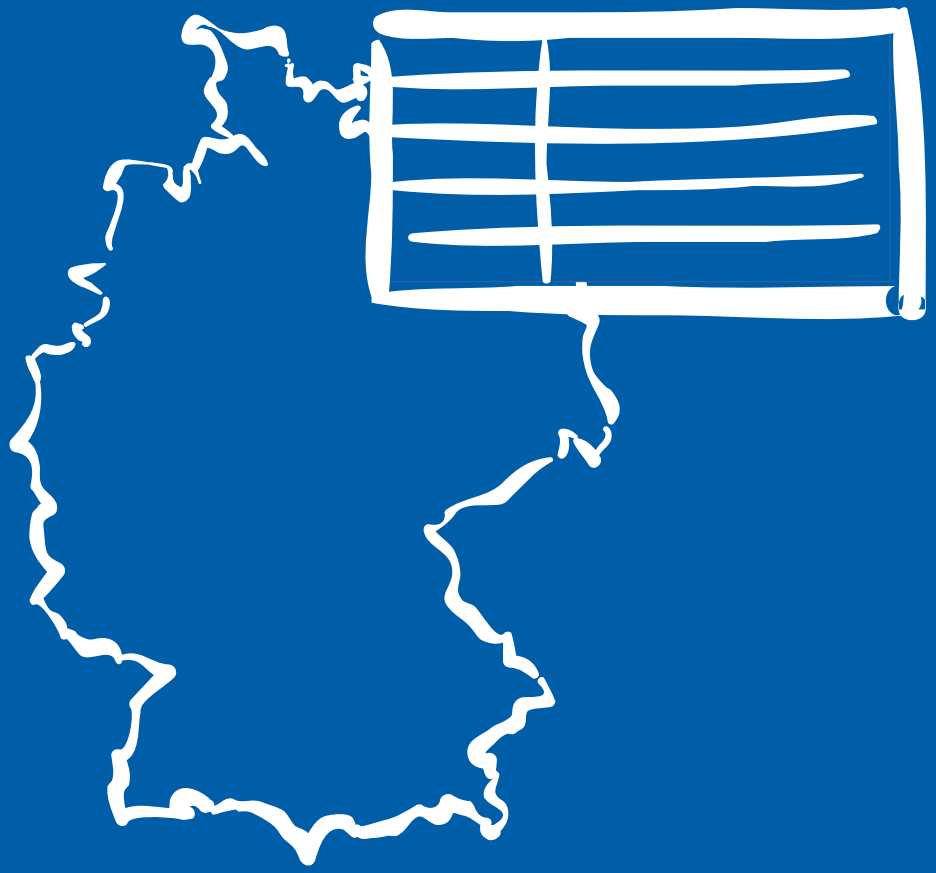
Webservice für Geoobjekte





- Öffentliche Schnittstelle
- Verwendung Standards
- In Zukunft einige Projekte mit JSON und Geoobjekten

Geoobjekt



feature

```
{
 "type": "Feature",
 "crs" : "EPSG:31467",
 "geometry": {
   "type" : "Polygon",
   "coordinates" : [
           [ [ 3479727.96, 5990327.04 ]
           , [ 3479528.96, 5990445.04 ]
           , [ 3479146.96, 5990823.04 ]
                       :
           , [ 3479727.96, 5990327.04]
            ]
          ]
 },
 "properties": {
    "name": "Deutschland",
    "capitol" : "Berlin",
    "area" : 357375.62,
    "citizens" : 82175684
 }
```

Geometrien

Point

Point

```
{ "type": "Point",
   "coordinates": [102.0, 2.0]
}
```

Geometrien

Line

Line

```
{ "type": "LineString",
   "coordinates": [
      [102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0]
    ]
}
```

Geometrien

Polygon

Polygon

```
"type": "Polygon",
  "coordinates":  [[
      [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]
    ] ]
   ]
```

Geometrien

- *Point*
- *Line*
- *Polygon*
- *Multi Point*
- *Multi Line*
- *Multi Polygon*
- *Geometry Collection*

GeometryCollection

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

Beispiel: Service - whos on first

whosonfirst-data / **whosonfirst-data**

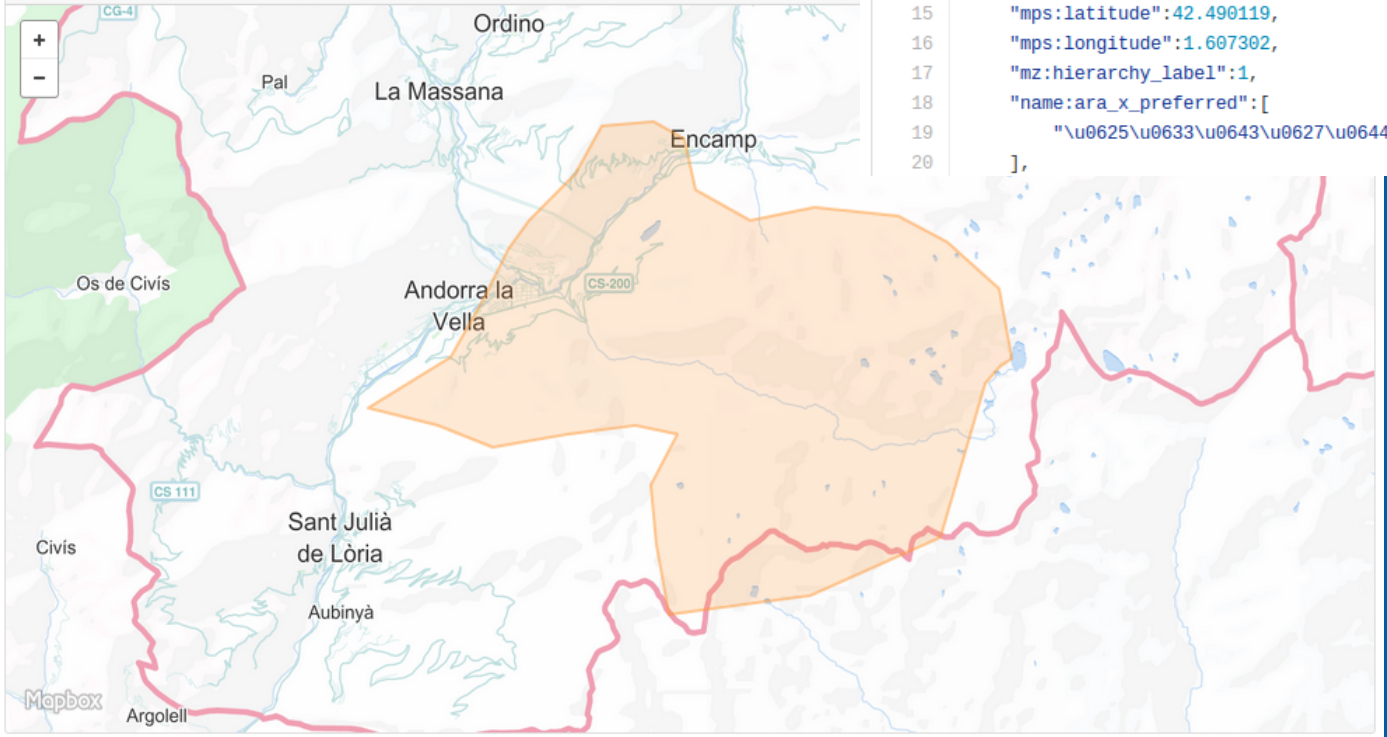
Code | Issues 224 | Pull requests 31 | Projects 0 | Pulse

Branch: **master** ▾ **whosonfirst-data** / data / 856 / 679 / 41 / **85667941.geojson**

thisisaaronland update names per issue #346

1 contributor

221 lines (221 sloc) | 7.21 KB



221 lines (221 sloc) | 7.21 KB


```
{
  "id": 85667941,
  "type": "Feature",
  "properties": {
    "edtf:cessation":"uuuu",
    "edtf:inception":"uuuu",
    "geom:area":0.007252,
    "geom:bbox":"1.49908936968,42.4530748538,1.65698368305,42.5388574147",
    "geom:latitude":42.495963,
    "geom:longitude":1.588761,
    "gn:population":16391,
    "iso:country":"AD",
    "lbl:latitude":42.490119,
    "lbl:longitude":1.607302,
    "mps:latitude":42.490119,
    "mps:longitude":1.607302,
    "mz:hierarchy_label":1,
    "name:ara_x_preferred":[
        "\u0625\u0633\u0643\u0627\u0644\u062f\u064a\u0633 \u0623\u0646\u062c\u0648\u0631\u0
    ],
```

Webservice für Geoobjekte - Ziele

- *Verwendung von Standards*
- *Schema zur Dokumentation der Datentypen*
- *JSON als Datenstruktur*

Idee GEOJSON

- *Quasi Standard seit 2008*
- *Seit diesem Jahr Internet Engineering Task Force (IETF) - Standard RFC 7946*

GEOJSON Schema

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "id": "http://json-schema.org/geojson/geometry.json#",
   "title": "geometry",
   "description": "One geometry as defined by GeoJSON",
   "type": "object",
   "required": [ "type", "coordinates" ],
   "oneOf": [
      {
         "title": "Point",
         "properties": {
            "type": { "enum": [ "Point" ] },
            "coordinates": { "$ref": "#/definitions/position" }
         }
      },
      {
         "title": "MultiPoint",
         "properties": {
            "type": { "enum": [ "MultiPoint" ] },
            "coordinates": { "$ref": "#/definitions/positionArray" }
```

Standardvorgehen

1. *XSD-Schema definieren*
2. *Klassen über JAXB generieren*
3. *JSON mit Jackson lesen/schreiben*

Geometry.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
 jaxb:version="2.0" targetNamespace="http://www.anwiba.net/geometry"
 xmlns:this="http://www.anwiba.net/geometry"
 elementFormDefault="qualified">
 <xsd:annotation>
  <xsd:appinfo>
   <jaxb:schemaBindings>
    <jaxb:package name="net.anwiba.generated.geometry" />
   </jaxb:schemaBindings>
  </xsd:appinfo>
 </xsd:annotation>

 <xsd:element name="geometry" type="this:GeometryType">
 </xsd:element>

 <xsd:complexType name="GeometryType">
  <xsd:annotation>
   <xsd:appinfo>
    <jaxb:class name="Geometry"/>
   </xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
   <xsd:element name="crs" type="xsd:string" minOccurs="0" maxOccurs="1">
   </xsd:element>
   <xsd:element name="type" type="xsd:string" minOccurs="1" maxOccurs="1" default="Geome..
   </xsd:element>
   <xsd:element name="coordinates" type="this:CoordinatesType" minOccurs="1" maxOccurs=...
   </xsd:element>
```

Problem

multidimensionale Array's

Polygon

```
coordinates: [
      [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]
     ]
    ]
```

Definition so, dass JAXB und Jackson funktinonieren

Geometry.xsd

```
<xsd:complexType name="CoordinatesType">
 <xsd:annotation>
  <xsd:appinfo>
   <jaxb:class name="Coordinates"/>
  </xsd:appinfo>
 </xsd:annotation>
  ?
</xsd:complexType>
```

Welche Alternativen

1 - JSON-Schema und jsonschema2pojo

2 - Kein Schema und Java Klassen mit Annotationen für Serialiser implementieren

3 - Kein JSON sondern XSD Schema und XML (GML)

Alternative 1 - JSON-Schema und jsonschema2pojo

- ✔ *JSON*
- ✔ *Standard*
- ✔ *Schema als Dokumentation*

- ☹ *Schwer lesbar*
- ☹ *Jsonschema2pojo*
  - *oneOf, allOf, anyOf nicht umgesetzt*

GEOJSON Schema

```
"definitions": {
  "position": {
    "description": "A single position",
    "type": "array",
    "minItems": 2,
    "items": [
      { "type": "number" },
      { "type": "number" }
    ],
    "additionalItems": false
  },
  "positionArray": {
    "description": "An array of positions",
    "type": "array",
    "items": { "$ref": "#/definitions/position" }
  },
  "lineString": {
    "description": "An array of two or more positions",
    "allOf": [
      { "$ref": "#/definitions/positionArray" },
      { "minItems": 2 }
    ]
  },
  "linearRing": {
```

Alternative 2 - Kein Schema und Java Klassen implementieren

- ✔ *JSON*
- ✔ *Standard*
- ✔ *Wäre für Projekt ausreichend*

- ☹ *Kein Schema*
- ☹ *Schwer Lesbar*
- ☹ *Unproduktiv*
  bei hoher Anzahl von Java Klassen

Geometry.java

```
//Copyright (c) 2016 by Andreas W. Bartels
package net.anwiba.gis.geo.json.v01_0;

import com.fasterxml.jackson.annotation.JsonProperty;

// Geo JSON feature object
// @see: https://tools.ietf.org/html/rfc7946#section-3.2
public class Feature {

   private String type = "Feature";
   private Object id = null;
   private Crs crs = null;
   private double[] bbox = null;
   private Geometry geometry = null;
   private Properties properties = null;

   @JsonProperty("type")
   public String getType() {
      return this.type;
   }

   @JsonProperty("id")
   public void setId(final Object id) {
      this.id = id;
```

Alternative 3 - Kein JSON mit XML basierter Lösung

- ✔ *Verwendung von Standard*
- ✔ *Dokumentation in über Schema Dateien*
- ✔ *Wäre für Projekt ausreichend*

- ☹ *Kein JSON*
- ☹ *Schwer Lesbar*
- ☹ *Unproduktiv*
  bei hoher Anzahl von Datentypen

GML Schema

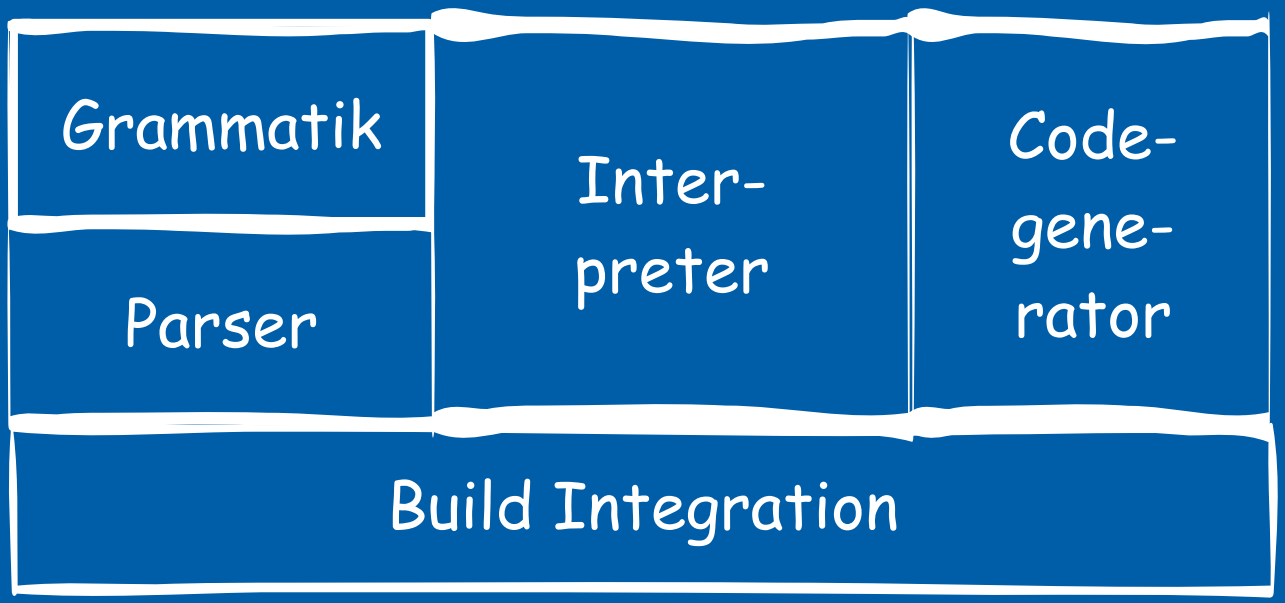
```
<!-- ============================================================
     There are two ways to represent coordinates: (1) as a sequence
     of <coord> elements that encapsulate tuples, or (2) using a
     single <coordinates> string.
  ============================================================ -->
        <complexType name="CoordType">
                <annotation>
                        <documentation>
     Represents a coordinate tuple in one, two, or three dimensions.
    </documentation>
                </annotation>
                <sequence>
                        <element name="X" type="decimal"/>
                        <element name="Y" type="decimal" minOccurs="0"/>
                        <element name="Z" type="decimal" minOccurs="0"/>
                </sequence>
        </complexType>
        <complexType name="CoordinatesType">
                <annotation>
                        <documentation>
     Coordinates can be included in a single string, but there is no
     facility for validating string content. The value of the 'cs' attribute
     is the separator for coordinate values, and the value of the 'ts'
     attribute gives the tuple separator (a single space by default); the
     default values may be changed to reflect local usage.
    </documentation>
                </annotation>
                <simpleContent>
```

Zwischen Bilanz

- *Im Projekt nutzten wir GML also XML-basiert*

- *Ziel-Verwendung von JSON und Schemas*
  - *Schnelles definieren von Datentypen anhand von JSON-Ausprägungen (Produktivität)*
  - *Umgang mit Geoobjekten (Standards GEOJSON/ARC GIS REST (OGC))*
  - *Dokumentation der Datentypen über Schema-Dateien*
  - *Schema (Übersichtlich, leicht Verständlich)*

Konzept und Evaluierung



- *Parser und Grammatik – antlr*
- *Code Generator – JAXB-Codegen*
- *Build Integration - Ant Task und Maven Plugin*
- *JSON lesen/schreiben - Jackson*

Object

foo.jssd

```
{}
```

Foo.java

```
//Copyright (c) 2016 by Andreas W. Bartels
package net.anwiba.json.schema.example;


public class Foo {


}
```

Members

bar.jssd

```
{
  "member": <string> null
}
```

Bar.java

```
//Copyright (c) 2016 by Andreas W. Bartels
package net.anwiba.json.schema.example;

import com.fasterxml.jackson.annotation.JsonProperty;

public class Bar {

    private String member = null;

    @JsonProperty("member")
    public void setMember(final String member) {
        this.member = member;
    }

    @JsonProperty("member")
    public String getMember() {
        return this.member;
    }

}
```

Binding

bar.jssd

```
{
  "type": <String> "Bar",
  "foos": <foo[]> null
}
```

Bar.java

```
:
public class Bar {
    private String type = "Bar";
    private Foo[] foos = null;

    @JsonProperty("type")
     public void setType(final String type) {
       this.type = type;
    }
    @JsonProperty("type")
    public String getType() {
       return this.type;
    }
    @JsonProperty("foos")
     public void setFoos(final Foo[] foos) {
       this.foos = foos;
    }
    @JsonProperty("foos")
    public Foo[] getFoos() {
       if (foos == null) {
         return new Foos[0];
       }
       return this.foos;
    }
```

Dokumentation

bar.jssd

```
/*
* Beispiel Struktur
*/
{
  // type bar
  "type": <String> "Bar",
  // array of foos
  "foos": <foo[]> null
}
```

Bar.java

```
//Copyright (c) 2016 by Andreas W. Bartels
package net.anwiba.json.schema.example;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Bar {
    private String type = "Bar";
    private Foo[] foos = null;

    @JsonProperty("type")
    public void setType(final String type) {
        this.type = type;
    }
     :
    @JsonProperty("foos")
    public void setFoos(final Foo[] foos) {
        this.foos = foos;
    }
    @JsonProperty("foos")
    public Foo[] getFoos() {
        if (foos == null) {
          return new Foos[0];
        }
        return this.foos;
    }
```

Verhalten

bar.jssd

```
/*
* Beispiel Struktur
*/
{
  // type bar
  "type": <String> "Bar",
  // array of foos
  "foos": <foo[]> null
}
```

Bar.java

```
//Copyright (c) 2016 by Andreas W. Bartels
package net.anwiba.json.schema.example;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Bar {
   private String type = "Bar";
   private Foo[] foos = null;

   @JsonProperty("type")
   public void setType(final String type) {
      this.type = type;
   }
    :
   @JsonProperty("foos")
   public void setFoos(final Foo[] foos) {
      this.foos = foos;
   }
   @JsonProperty("foos")
   public Foo[] getFoos() {
      if (foos == null) {
        return new Foos[0];
      }
      return this.foos;
   }
```

Verhalten

bar.jssd

```
@JssdUnknownMember
{
  "type": <String> "Bar",
  "foos": <foo[]> null
}
```

Bar.java

```
//Copyright (c) 2016 by Andreas W. Bartels
package net.anwiba.json.schema.example;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Bar {
   private String type = "Bar";
   private Foo[] foos = null;
   private final Map<java.lang.String, Object> _unknownMembers
      = new LinkedHashMap<java.lang.String, Object>();

    @JsonAnySetter
    public void set(final java.lang.String name, final Object value) {
        Ensure.ensureThatArgument(name, Conditions.notNull());
        this._unknownMembers.put(name, value);
    }
    @JsonAnyGetter
    public Map<java.lang.String, Object> get() {
        if (this._unknownMembers.isEmpty()) {
            return null;
        }
        return this._unknownMembers;
    }

   @JsonProperty("type")
```

GEOJSON

Was noch fehlt ist Vererbung

GEOJSON Schema

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "id": "http://json-schema.org/geojson/geometry.json#",
   "title": "geometry",
   "description": "One geometry as defined by GeoJSON",
   "type": "object",
   "required": [ "type", "coordinates" ],
   "oneOf": [
      {
         "title": "Point",
         "properties": {
            "type": { "enum": [ "Point" ] },
            "coordinates": { "$ref": "#/definitions/position" }
         }
      },
      {
         "title": "MultiPoint",
         "properties": {
            "type": { "enum": [ "MultiPoint" ] },
            "coordinates": { "$ref": "#/definitions/positionArray" }
```

Vererbung

geometry.jssd

```
// title: Geo JSON geometry ob...
// description: Schema for a Ge...
// reference: https://tools.ietf...
[

 // see: https://tools.ietf...
 @JssdName(value="geometry")
 {
  "type" : <String> "Geometry",
  "crs": <crs> null,
  "bbox": <double[]> null
 },

 // see: https://tools.ietf...
 @JssdName(value="point")
 @JssdExtends(type="geometry"
 {
  "type": <String> "Point",
  @JssdNotNullable
  "coordinates": <Number[]> null
```

Point.java

```
//Copyright (c) 2016 by Andreas W. Bartels
package net.anwiba.json.schema.example;
import com.fasterxml.jackson.annotation.JsonProperty;
import net.anwiba.commons.ensure.Conditions;
import net.anwiba.commons.ensure.Ensure;

public class Point extends Geometry {
   private String type = "Point";
   private Number[] coordinates = null;

   @JsonProperty("type")
   public void setType(final String type) {
      this.type = type;
   }

   @JsonProperty("type")
   public String getType() {
      return this.type;
   }

   @JsonProperty("coordinates")
   public void setCoordinates(final Number[] coordinates) {
      Ensure.ensureThatArgument(coordinates, Conditions.notNull());
      this.coordinates = coordinates;
```

Vererbung

Automatisches
erkennen
des Datentypes

feature.geojson

```
{
 "type": "Feature",
 "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
 },
 "properties": {
  "name": "Dinagat Islands"
 }
}
```

Vererbung

geometry.jssd

```
// title: Geo JSON geometry ob...
// description: Schema for a Ge...
// reference: https://tools.ietf...
[

 @JssdName(value="geometry")
 @JssdFactory
 {
  "type" : <String> "Geometry",
  "crs": <crs> null,
  "bbox": <double[]> null
 },

 @JssdName(value="point")
 @JssdExtends(type="geometry")
 {
  "type": <String> "Point",
  @JssdNotNullable
  "coordinates": <Number[]> null
 },
```

Geometry.java

```
import net.anwiba.commons.utilities.string.StringUtilities;

public class Geometry {

   private String type = "Geometry";
   private Crs crs = null;
   private double[] bbox = null;

@JsonCreator
public static Geometry create(
    @JsonProperty("type") String type) {
  if (StringUtilities.isNullOrTrimmedEmpty(type)) {
   return new Geometry();
  }
  Class<? extends Geometry>clazz=_createClass(type);
  if (clazz!= null) {
   return _createBean(clazz);
  }
  return new Geometry();
}
```

Typ-Sicherheit

RFC 7946 – GEOJSON

3.2. Feature Object

O 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.**

☹ Member mit Variierenden Datentypen

java.lang.Object als Typ möglich, ist es aber nicht befriedigend.

Maven Integration

pom.xml

```
<plugin>
  <groupId>net.anwiba.commons.tools</groupId>
  <artifactId>anwiba-tools-definition-schema-json</artifactId>
  <version>1.0.0</version>
  <executions>
    <execution>
      <id>geojson</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>generate</goal>
      </goals>
      <configuration>
        <pakkage>net.anwiba.commons.schema.geojson.v1_00</pakkage>
        <comment>Copyright (c) 2016 by Andreas W. Bartels</comment>
      </configuration>
    </execution>
```

Java Integration

Snippet.java

```
import com.fasterxml.jackson.databind.ObjectMapper;
import net.anwiba.gis.geo.json.v01_0.Geometry;


// json Struktur lesen
String body = "{\"type\": \"Point\", \"coordinates\": [125.6, 10.1]}";
Geometry geometry = new ObjectMapper().
                                readerFor(Geometry.class).
                                readValue(body);


// nach json Struktur schreiben
new ObjectMapper()
      .writerFor(Geometry.class)
      .writeValue(outputStream, geometry);
```

Mögliche Weiterentwicklungen:

- *Umgang mit Versionen*
  (Integration Transformation zwischen Versionen)
- *Kommentare in Generierten Source Code übernehemen*
- *Generator Json Ausprägung -> JSSD*
- *Members mit Variierende Typen, ev. über genersiche Methoden*
- *Transformator JSON - Schema -> JSSD -> JSON - Schema*

Fazit - Ziele

☺ *Schnelles Definieren von Datentypen anhand von JSON-Ausprägungen*

☺ *Umgang mit Geoobjekten (Standards GEOJSON/ARC GIS REST (OGC))*

☺ *Dokumentation der Datentypen über Schema-Dateien*

☹ *Umgang mit Variierende Datentypen*

☹ *Keine Übernahme der Dokumentation in den generierten Code*

Fazit - Verwendungen

- ✔ *JGISShell*
  - *ARCGIS Rest Service Client, MAPBOX-Service Client, GEOJSON, GPSD-Client*
- ✔ *Cadenza Web und Desktop*
  - *ARCGIS Rest Service Client, MAPBOX-Service Client, CARTO-Service Client, GEO-Processing Service, etc.*
- ✔ *Cadenza Mobile*
  - *Daten austausch JSON-Schema (Draft 1,nicht vollständig)*
  - *Konfigurieren von Fachanwendungen*

jetzt veröffentlicht.

Referenzen

Link JSSD-Tool
https://github.com/AndreasWBartels/libraries/wiki/JSSD

link JGISShell

https://github.com/AndreasWBartels/JGISShell

Link Cadenza

http://www.disy.net/produkte/cadenza.html

Links zu besprochenen Themen

http://geojson.org/

http://json-schema.org/

https://github.com/joelittlejohn/jsonschema2pojo

http://www.opengeospatial.org/standards/gml


Danke
Fragen ?
Anregungen ?
Anmerkungen ?
Pulau Bangka
© Andreas W. Bartels