

Lesson 4.2

Semantic 3D city models

GE01004:
3D modelling of the built environment

<https://3d.bk.tudelft.nl/courses/geo1004>



3D geoinformation

Department of Urbanism
Faculty of Architecture and the Built Environment
Delft University of Technology

JSON

- **JavaScript Object Notation**
- Lightweight data-interchange format
- It is easy for humans to read and write
- It is text-based
- It is easy for machines to parse and generate
- It is originally based on a subset of JavaScript (but is now in most languages)

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Virtually all languages have native support for it

It is built on two (very common) **data structures**:

1. collection of name-value pairs
 - Python dictionaries (`a={}`)
 - C++ `std::map`
 - also called a “hash” or “hashmap”
2. arrays
 - Python lists (`a=[]`)
 - C++ `std::vector` / `std::array`

Data types:

- number (float and double)
- integer
- string
- Boolean
- null
- *(no support for datetime type)*

To represent a person

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Key-value pair

"name": "Hugo"

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "1.1",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  },
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

CityJSON

CityGML-XML files are very complex

- files are deeply nested, and large
- many “points of entry”
- many diff ways to do one thing

- ➔ few software packages use CityGML
- ➔ no parsers in JavaScript
- ➔ I personally get 😞 each time I get a new file



MSc Geomatics students when trying to parse a CityGML file in Python



version 1.1 = current version

2.0 (since September 2023)



official community standard

- compliant with CityGML v3.0
- subset of CityGML (~90% of features)
- software for full conversion CityGML <-> CityJSON
- several software support CityJSON

1st-level City Objects

Bridge
Building
CityFurniture
CityObjectGroup
GenericCityObject
LandUse
OtherConstruction
PlantCover
SolitaryVegetationObject
TINRelief
TransportationSquare
Railway
Road
Tunnel
WaterBody
Waterway
+Extension

2nd-level City Objects

BridgePart
BridgeInstallation
BridgeConstructiveElement
BridgeRoom
BridgeFurniture

BuildingPart
BuildingInstallation
BuildingConstructiveElement
BuildingFurniture
BuildingStorey
BuildingRoom
BuildingUnit

TunnelPart
TunnelInstallation
TunnelConstructiveElement
TunnelHollowSpace
TunnelFurniture

Specifications give all the gory details

CityJSON Specifications 2.0.1

Living Standard, 11 April 2024

This version:
<https://cityjson.org/specs/2.0.1/>

Latest published version:
<https://cityjson.org/specs/>

Previous Versions:
<https://cityjson.org/specs/overview/>

Feedback:
[GitHub](#)

Editors:
[Hugo Ledoux](#) (TU Delft)
[Balázs Dukai](#) (3DGI)

© The editors have waived all copyright and related or neighbouring rights to this work. The CityJSON Specifications are marked with [CC0 1.0 Universal](#).

Abstract

CityJSON is a data exchange format for digital 3D models of cities and landscapes. It aims at being easy-to-use (for reading, processing, and creating datasets), and it was designed with programmers in mind, so that tools and APIs supporting it can be quickly built. The [JSON](#)-based encoding of CityJSON implements a subset of the [OGC CityGML data model \(version 3.0\)](#) and includes a JSON-specific extension mechanism. Using JSON instead of GML allows us to compress files by a factor 6 and at the same time to simplify greatly the structure of the files.

TABLE OF CONTENTS

- 1 **CityJSON Object**
- 2 **The different City Objects**
 - 2.1 Attributes for all City Objects
 - 2.2 Bridge
 - 2.3 Building
 - 2.4 CityFurniture
 - 2.5 CityObjectGroup
 - 2.6 GenericCityObject
 - 2.7 LandUse
 - 2.8 OtherConstruction
 - 2.9 PlantCover
 - 2.10 SolitaryVegetationObject
 - 2.11 TINRelief
 - 2.12 Transportation
 - 2.13 Tunnel
 - 2.14 WaterBody
- 3 **Geometry Objects**
 - 3.1 Coordinates of the vertices
 - 3.2 Arrays to represent boundaries
 - 3.3 Semantics of geometric primitives
 - 3.4 Geometry templates
- 4 **Transform Object**
- 5 **Metadata**

A CityJSON file

```
{  
  "type": "CityJSON",  
  "version": "2.0",  
  "metadata": {  
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"  
  },  
  "transform": {...}  
  "CityObjects": {  
    "id-1": {  
      "type": "Building",  
      "attributes": {  
        "measuredHeight": 22.3,  
        "roofType": "gable",  
        "owner": "Elvis Presley"  
      },  
      "geometry": [  
        {  
          "type": "MultiSurface",  
          "boundaries": [  
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]  
          ]  
        }  
      ]  
    }  
  }  
},  
  "vertices": [  
    [231, 23212, 110],  
    [1111, 3211, 120],  
    ...  
  ],  
  "appearance": {  
    "materials": [],  
    "textures": [],  
    "vertices-texture": []  
  }  
}
```

version CityJSON

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    },
    ...
  ],
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

metadata possible, eg:

- CRS
- Point of contact
- bbox of dataset

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSSG/0/7415"
  },
  "transform": [...],
  "CityObjects": [
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    }
  ],
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

All City Objects listed here, *indexed* by their ID

Each have geometries + attributes

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    },
    "vertices": [
      [231, 23212, 110],
      [1111, 3211, 120],
      ...
    ],
    "appearance": {
      "materials": [],
      "textures": [],
      "vertices-texture": []
    }
  }
}
```

Geometry is ID of the vertex, global list
compression + more "topology"

A CityJSON file

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSSG/0/7415"
  },
  "transform": {...}
  "CityObjects": {
    "id-1": {
      "type": "Building",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    },
    ...
  ],
  "vertices": [
    [231, 23212, 110],
    [1111, 3211, 120],
    ...
  ],
  "appearance": {
    "materials": [],
    "textures": [],
    "vertices-texture": []
  }
}
```

material + texture possible

BuildingParts: links between City Objects

```
"CityObjects": {  
  "id-1": {  
    "type": "Building",  
    "attributes": {  
      "roofType": "gable"  
    },  
    "children": ["id-56", "id-832", "mybalcony"]  
  },  
  "id-56": {  
    "type": "BuildingPart",  
    "parents": ["id-1"],  
    ...  
  },  
  "mybalcony": {  
    "type": "BuildingInstallation",  
    "parent": ["id-1"],  
    ...  
  }  
}
```

BuildingParts: links between City Objects

```
"CityObjects": {  
  "id-1": {  
    "type": "Building",  
    "attributes": {  
      "roofType": "gable"  
    },  
    "children": ["id-56", "id-832", "mybalcony"]  
  },  
  "id-56": {  
    "type": "BuildingPart",  
    "parents": ["id-1"],  
    ...  
  },  
  "mybalcony": {  
    "type": "BuildingInstallation",  
    "parents": ["id-1"],  
    ...  
  }  
}
```

goal == a flat schema

CityJSON software

web-viewer: ninja.cityjson.org

The screenshot shows the CityJSON Ninja web viewer interface. The browser address bar displays <https://ninja.cityjson.org/#>. The page title is "CityJSON Ninja". The interface includes a search bar for IDs, object types, or attributes, and buttons for "Download" and "Close". A list of "City Objects" (2498 total) is shown, with the selected object being `GUID_FBC35DA1-A388-4EA3-A4EA-68703A790603`. The detailed view for this building shows 5 attributes and 1 geometry. The attributes table is as follows:

Attribute	Value
roofType	1000
RelativeEavesHeight	9.833
RelativeRidgeHeight	9.833
AbsoluteEavesHeight	16.181
AbsoluteRidgeHeight	16.181

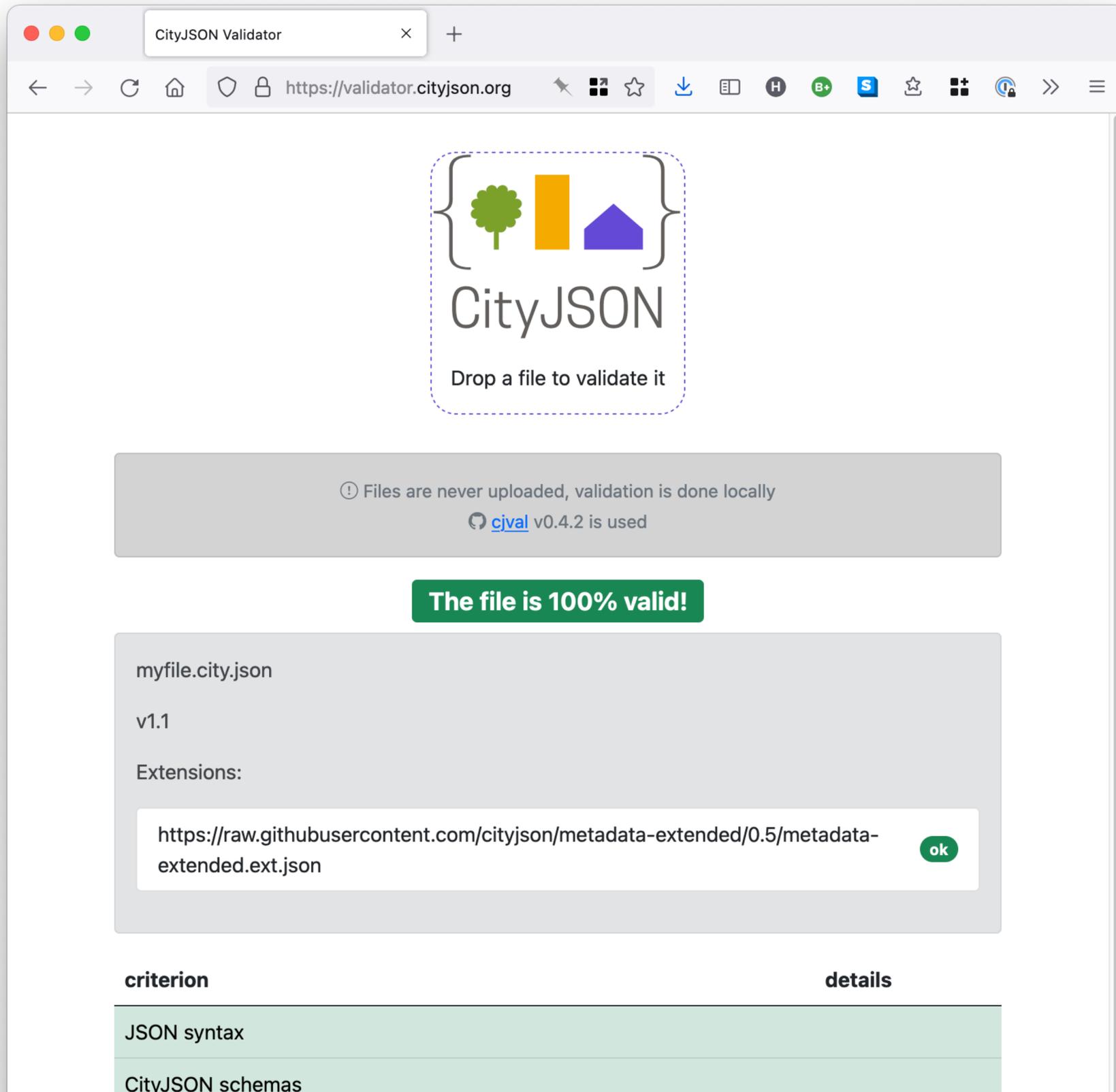
The background shows a 3D visualization of a city model with blue buildings on a yellow ground plane.

Nice fact: programmed mostly by 2 Geomatics students, as projects for GEO5010

You can do the same (or similar!)

Validation of the syntax of a file: cjval

A GE05010 project from a student also kick-started this!



The screenshot shows the CityJSON Validator interface. At the top, there is a logo with a tree, a bar chart, and a house, with the text "CityJSON" and "Drop a file to validate it". Below the logo, a grey box contains the text "Files are never uploaded, validation is done locally" and "cjval v0.4.2 is used". A green box in the center says "The file is 100% valid!". Below this, the file name "myfile.city.json" and version "v1.1" are shown. An "Extensions:" section contains a text box with the URL "https://raw.githubusercontent.com/cityjson/metadata-extended/0.5/metadata-extended.ext.json" and an "ok" button. At the bottom, there are two tabs: "criterion" and "details". Under "criterion", there are two items: "JSON syntax" and "CityJSON schemas".

```
{
  "type": "CityJSON",
  "version": "2.0",
  "metadata": {
    "referenceSystem": "https://www.opengis.net/def/crs/EPSG/0/7415"
  },
  "transform": {...},
  "CityObjects": {
    "id-1": {
      "type": "Buildnig",
      "attributes": {
        "measuredHeight": 22.3,
        "roofType": "gable",
        "owner": "Elvis Presley"
      },
      "geometry": [
        {
          "type": "MultiSurface",
          "boundaries": [
            [[0, 3, 2, 1]], [[4, 5, 6, 7]], [[0, 1, 5, 4]]
          ]
        }
      ]
    },
    "vertices": [
      [231, 23212, 110.223],
      [1111, 3211, 120],
      ...
    ],
    "appearance": {
      "materials": [],
      "textures": [],
      "vertices-texture": []
    }
  }
}
```

Python parser is simple



```
import json

fin = open('mycity.json')
cm = json.loads(fin.read())

print "There are", len(cm['City0bjects']), "City0bjects"

# list all ids
for id in cm['City0bjects']:
    print "\t", id
```

cjio (CityJSON/io) => pip install cjio

```
2. bash
Hugos-MacBook-Pro:rotterdam hugo$ cjio
Usage: cjio [OPTIONS] INPUT COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

Process and manipulate a CityJSON file, and allow different outputs. The
different operators can be chained to perform several processing in one
step, the CityJSON model goes through the different operators.

To get help on specific command, eg for 'validate':

    cjio validate --help

Usage examples:

    cjio example.json validate
    cjio example.json remove_textures info
    cjio example.json subset --id house12 remove_materials save out.json

Options:
  --version          Show the version and exit.
  --off              Load an OFF file and convert it to one CityJSON
                    GenericCityObject.
  --ignore_duplicate_keys Load a CityJSON file even if some City Objects have
                    the same IDs (technically invalid file)
  --help            Show this message and exit.

Commands:
  compress          Compress a CityJSON file, ie stores its...
  decompress        Decompress a CityJSON file, ie remove the...
  info              Output info in simple JSON.
  merge             Merge the current CityJSON with others.
  remove_duplicate_vertices Remove duplicate vertices a CityJSON file.
  remove_materials  Remove all materials from a CityJSON file.
  remove_orphan_vertices Remove orphan vertices a CityJSON file.
  remove_textures   Remove all textures from a CityJSON file.
  save              Save the CityJSON to a file.
  subset           Create a subset of a CityJSON file.
  update_bbox       Update the bbox of a CityJSON file.
  update_crs        Update the CRS with a new value.
  validate          Validate the CityJSON file: (1) against its...
```

```
$ cjio myfile.json crs_assign 7415 subset --cotype Building lod_filter 2.2 save out.json
```

No need to save temp files between operators: pipeline is used

QGIS plugin

The screenshot shows the QGIS desktop environment. The main window is titled "Untitled Project - QGIS" and displays a toolbar, a Layers panel on the left, and a central canvas area. A "News" window is open over the canvas, featuring a Ukrainian flag and a message titled "QGIS for Peace". In the foreground, the "Plugins | All (1052)" dialog box is open, displaying a search for "cityjson". The "CityJSON Loader" plugin is selected and its details are shown, including a description, ratings, and version information.

QGIS for Peace

A message of peace from the QGIS Community: We, the developers, contributors and community members of the QGIS Project view the ongoing world events in Ukraine and other conflict areas around the world with great sadness. Our aim in developing QGIS has always been to provide a powerful tool to support the creation of a just and humane society. We want to enable a world where every person has a voice, the ability to express, and be secure in their tenure in their homes, villages, towns, cities and countries. We hope tools like QGIS are used to the benefit of all citizens on earth, to support a sustainable environment, an orderly society and, in particular, to establish and preserve sovereign dignity, security and freedom from oppression.

CityJSON Loader

This plugin allows for CityJSON files to be loaded in QGIS

This plugin allows QGIS to load CityJSON datasets. Data are loaded in respective tables and all information are loaded.

★★★★★ 11 rating vote(s), 17723 downloads

Category Vector

Tags [python](#), [cityjson](#), [3d](#)

More info [homepage](#) [bug tracker](#) [code repository](#)

Author [3d geoinformation group \(TU Delft\)](#)

Installed version 0.8.0

Available version (stable) 0.8.0 updated at Mon Feb 7 05:49:27 2022

Changelog

- 0.8.0 - 7/2/2022
 - * Add support for all semantic surface attributes
- 0.7.4 - 27/1/2022
 - * Add support for CityJSON v1.1 (except for CityJSONFeature types)
- 0.7.3 - 11/1/2022

Buttons: Upgrade All, Uninstall Plugin, Reinstall Plugin

citygml4j/citygml4j: The Open Source Java API for CityGML

521 commits 1 branch 25 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

claussag 14 on Apr 20

citygml4j months ago

gradle/w a month ago

resources 2 months ago

src-gen/main/java added generated JAXB classes 3 months ago

src/main removed unnecessary properties from CityJSON input and output factories 2 months ago

.gitignore minor change 3 months ago

LICENSE changes license to Apache License, Version 2.0 2 years ago

README.md Update README.md 2 months ago

build.gradle updated gradle a month ago

gradlew using Gradle as build tool 4 months ago

gradlew.bat using Gradle as build tool 4 months ago

settings.gradle preparing release 2.7.0 2 months ago

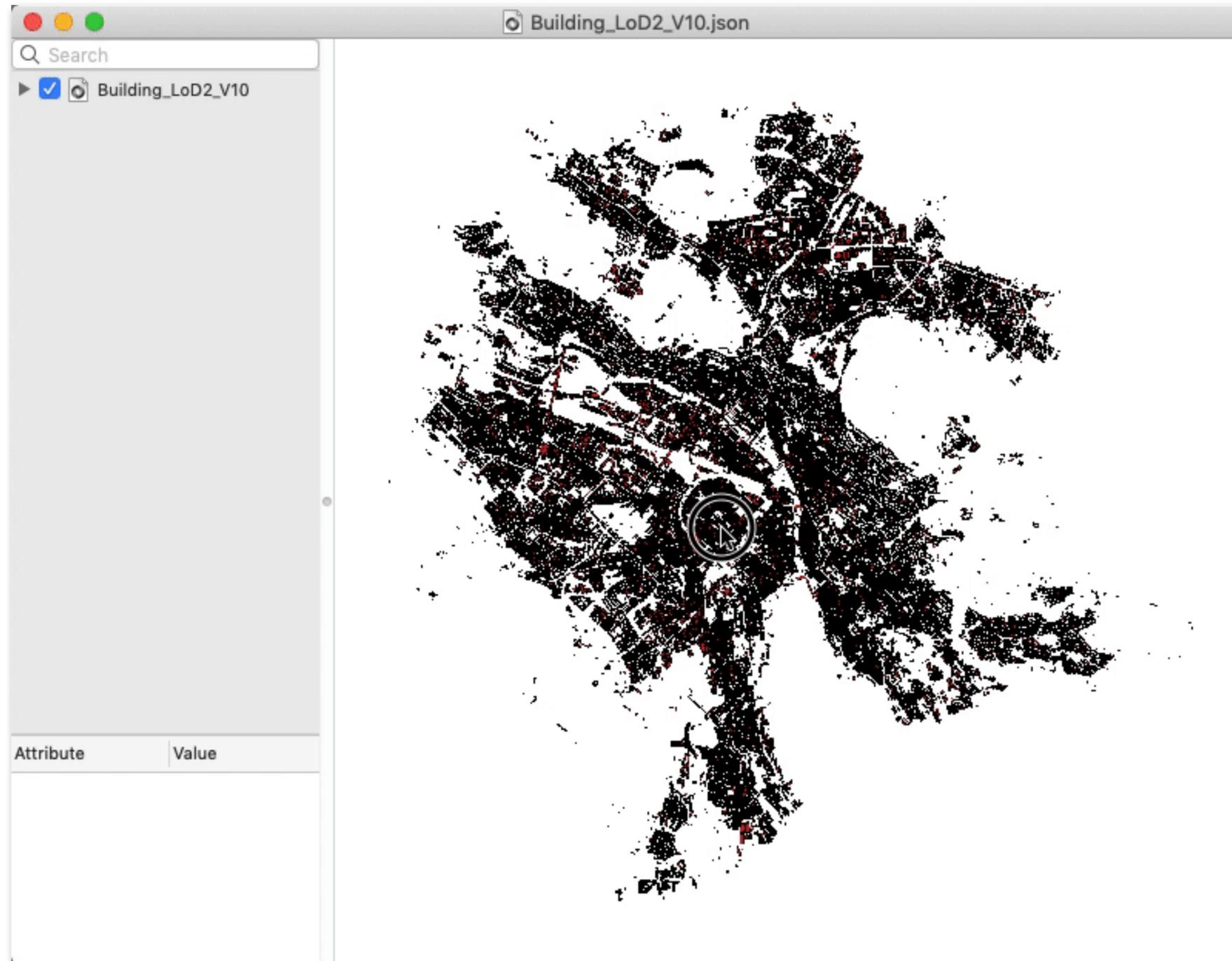
README.md

full conversion CityGML <-> CityJSON

Compression factor == ~6X

file	CityGML size (original)	CityGML size (w/o spaces)	textures?	CityJSON	CityJSON compressed	compression factor
CityGML demo "GeoRes"	4.3MB	4.1MB	yes	582KB	524KB	8.0
CityGML v2 demo "Railway"	45MB	34MB	yes	4.5MB	4.3MB	8.1
Den Haag "tile 01"	23MB	18MB	no, material	3.1MB	2.9MB	6.2
Montréal VM05	56MB	42MB	yes	5.7MB	5.4MB	7.8
New York LoD2 (DA13)	590MB	574MB	no	110MB	105MB	5.5
Rotterdam Delfshaven	16MB	15MB	yes	2.8MB	2.6MB	5.4
Vienna	37MB	36MB	no	5.6MB	5.3MB	6.8

One example: Zürich LoD2 buildings



CityGML = 3.0GB

(but 1GB of spaces/CRs/tabs!)

CityJSON = 292MB

Compression == 7.1X

Getting started?

Getting started with CityJSON | CityJSON

https://www.cityjson.org/tutorials/getting-started/

CityJSON

Search CityJSON

Tutorials / Getting started with CityJSON

Getting started with CityJSON

TABLE OF CONTENTS

- 1 [Download a simple file with 2 buildings](#)
- 2 [Visualise it](#)
- 3 [Manipulate and edit it with cjio](#)
- 4 [What else?](#)
- 5 [Questions and need help?](#)

Download a simple file with 2 buildings

Download [twobuildings.city.json](#), a simple file with 2 buildings.

You can open that file in any text editor to see its structure, and notice that you can manually edit it to change values and/or add new buildings, new metadata, or delete some attributes.

```
twobuildings.city.json
1 {
2   "type": "CityJSON",
3   "version": "1.1",
4   "metadata":
5     {
6       "geographicalExtent":
7         [
8           300578.235,
9           5041250.061,
10          13.688,
11          300618.138,
12          5041289.394,
13          29.45
14        ]
15      },
16   "CityObjects":
17     {
18       "Building_1":
19         {
20           "geometry":
21             [
22               {
23                 "boundaries":
24                   [
25                     [
26                       [
27                         0,
28                         1
```

Need help? Want to contribute?
Spotted an error?

Validation of a CityJSON file | CityJSON

https://www.cityjson.org/tutorials/validation/

CityJSON

Search CityJSON

Tutorials / Validation of a CityJSON file

Validation of a CityJSON file

TABLE OF CONTENTS

- 1 [Schema validation \(is the syntax of the file OK?\)](#)
- 2 [Geometry \(are the geometric primitives valid?\)](#)

Validation of a CityJSON dataset means that one must ensure that it respects the standardised specifications and definitions as given in the [specifications](#).

Schema validation (is the syntax of the file OK?)

The JSON schemas of CityJSON can be downloaded, for each version, at <https://www.cityjson.org/schemas/>. These are based on the [JSON Schema project](#).

To validate a given file you can use any software listed [here](#). However, it is rather tricky to stitch all the schemas together, and the handling of [Extensions](#) will not work.

The "official validator" for CityJSON is [cjval](#), which is [available as a web-app](#) and with [cjio](#).

To validate the file [twobuildings.city.json](#), simply drag it to <https://validator.cityjson.org>:

CityJSON Validator

https://validator.cityjson.org

Drop a file to validate it

(cjval v0.3.0 is used)
(files are never uploaded, validation is done locally)

The file is 100% valid!

twobuildings.city.json

Need help? Want to contribute?
Spotted an error?