**Community Standard Justification: 22-009**

**TITLE: CityJSON v1.1**

**CONTRIBUTOR**

**Name: Linda van den Brink**

**Organization: Geonovum**

**Address: Barchman Wuytierslaan 10, 3818LH Amersfoort, the Netherlands**

**Phone: +31 334604100**

**Email: l.vandenbrink@geonovum.nl**

**DATE: July 11th, 2022**

# Introduction

This document provides a justification to the OGC Technical Committee (TC) for consideration of CityJSON (v1.1) as a Community standard. This justification, along with the submitted candidate Community standard, will form the basis for TC review and vote to approve the start of a Work Item as the first step in the Community standard process for this standard.

The submitters agree to abide by the TC Policies and Procedures and OGC Intellectual Property Rights Policy ([http://www.opengeospatial.org/ogc/policies)](http://www.opengeospatial.org/ogc/policies%29) during the processing of this submission.

Once approved, the Community standard Work Item defined by this document is valid for six (6) months.

# Overview of proposed submission

*Summarize the proposed Community standard. In this summary, provide an overview of the geospatial interoperability requirements the proposed standard supports, the history of its development and use, and use cases.*

*(note: the text here has generally been taken from the official website https://cityjson.org)*

**Description**

CityJSON version 1.0 is a JSON-based encoding for a subset of the OGC CityGML data model (version 2.0.0), which is an open standardized data model and exchange format to store digital 3D models of cities and landscapes. CityJSON v1.0 has been accepted as a Community standard in August 2021.

CityJSON defines ways to describe, in a JSON encoding, most of the common 3D features and objects found in cities (such as buildings, roads, rivers, bridges, vegetation and city furniture) and the relationships between them. It also defines how to encode different standard levels of detail (LoDs) for the 3D objects in JSON, which allows us to represent different resolutions of objects for different applications and purposes.

A CityJSON file describes both the geometry and the semantics of the city features of a given area, e.g., buildings, roads, rivers, trees, and the city furniture. A CityJSON object, representing a city, is as ‘flat’ as possible, i.e., the hierarchy of CityGML has been flattened out and only the city objects which are ‘leaves’ of this hierarchy are implemented. This considerably simplifies the storage of a city model, and furthermore does not mean that information is lost.

The original justification for including CityJSON as a Community standard can be found at https://portal.ogc.org/files/91843. [Ledoux 2019] provides the arguments for developing CityJSON in full.

The current submission contains a proposal for updating the CityJSON v1.0 Community standard with the CityJSON specifications from version 1.1.1. The proposed update contains significant improvements over v1.0. Most notably, with this update the CityJSON Community standard will support the OGC CityGML data model version 3.0.

The developers of CityJSON consider CityJSON v1.1.1 to be an implementation of the CityGML v3.0 data model.

**Changes in CityJSON between v1.0 and v1.1.1**

Added:

* **CityJSONFeature** is a new JSON object type which is used for streaming large city models. Each CityJSONFeature is an individual CityObject, but unlike a CityJSON object, the vertices of a CityJSONFeature are local to the CityObject. CityJSONFeatures can be stored in a new line delimited text file or passed between processes as a data stream. When CityJSONFeatures are stored in a file, the file follows the JSON Lines specifications (<https://jsonlines.org/> ).
* **OtherConstruction** is a new CityObject type that is used for constructions that are not buildings, bridges or tunnels.
* **Interior elements and constructive elements of the Bridge, Building, Tunnel** objects are added.
	+ **Bridge**: BridgeRoom, BridgeFurniture
	+ **Building**: BuildingConstructiveElement, BuildingFurniture, BuildingStorey, BuildingRoom, BuildingUnit
	+ **Tunnel**: TunnelConstructiveElement, TunnelHollowSpace, TunnelFurniture
* **The semantic types *FloorSurface*, *InteriorWallSurface, CeilingSurface*** to the Building-related CityObject types.

Removed:

* **GenericCityObject** CityObject type has been moved from the core specifications to an Extension.
* **The attribute *trafficDirection*** has been removed.

Changed:

* **Transformation of vertices is mandatory,** thus the **Transform** object is a mandatory part of the CityJSON file. The vertices are transformed by scaling and translating them so that the resulting coordinates are stored as integers instead of floating point numbers. Transformation reduces the file size and it was already strongly recommended in previous versions.
* **Level of Detail is defined as a string instead of a number,** which allows easier parsing of the extended LoD values used by CityJSON.
* **Narrowed down the core Metadata properties** to six specific properties, such as *geographicalExtent*, *identifier*, *pointOfContact*, *referenceDate*, *referenceSystem*, *title*. The reduction was made to increase the clarity of the allowed metadata properties compared to the previous v1.0. Additional properties can be used and defined by an Extension, for instance the MetadataExtended extension <https://github.com/cityjson/metadata-extended> .
* **CityObject geometry is optional**, while in v1.0 it was mandatory.
* **CRS is defined with the OGC Name Type Specification**, for instance for EPSG:7415 is *https://www.opengis.net/def/crs/EPSG/0/7415,* while previously OGC CRS URNs were used. The new specification has the advantage that it is given in the form of a URL, potentially pointing to an online ressource.
* **CityObjectGroup lists the role of its children**, a property which was missing from v1.0.
* **Harmonised the geometry type restrictions for CityObjects** to improve the consistency across CityObject types.

**CityGML v3.0 compatibility**

CityJSON implements most of the data model of CityGML v3.0.0, and all the CityGML modules have been mapped to CityJSON objects. However, for the sake of simplicity and efficiency, some features have been omitted and/or simplified. The main features that are not supported in v1.1.1 are:

* **Several CRSs in the same datasets.** In CityJSON, all geometries in a given CityJSON object must use the same CRS.
* **Arbitrary coordinate reference systems (CRSs).** Only an [EPSG code](https://epsg.io/) can be used.
* **Identifiers for low-level geometries.** In CityGML most objects can have an ID (usually a `gml:id). That is, not only can one building have an ID, but also each of the 3D primitives forming its geometry can have an ID. In CityJSON, only city objects and semantic surfaces can have IDs.
* **Raster files for the relief.** Only TINs are supported.
* **CityGML class** GeoreferencedTexture. In the **Appearance** module, the CityGML class TexCoordGen is not supported, i.e. one must specify the UV coordinates in the texture files.
* **Topological relationships**, e.g. *relativeToTerrain* and *relativeToWater*, which qualify relationships, are not supported.
* **externalReference**, and **generalizesTo** properties of city objects
* CityObjectGroup **name**
* **Terrain Intersection Curve (TIC).** This feature of CityGML is seldom used in practice it seems, and can always be simply computed on-the-fly: intersection between the solid of the buildings (or other objects) and the terrain. Furthermore, it is dependent on the LoD of the object (different LoDs can have different footprints)
* **Complex attributes have been simplified**. For instance, several attributes in CityGML are derived from gml:Measure (like bldg:measuredHeight), and thus you cannot just store a value but also have to store the unit of measurement. This is not represented in CityJSON directly, an Extension must be used. Also, generic attributes in CityGML cannot be mapped simply because in CityJSON you can add any attributes you like (in line with the JSON philosophy).

A more complete list, including the mapping of the individual CityGML modules, is maintained at https://www.cityjson.org/conformance/v30/.

One feature that is wished by practitioners, but which is supplied in CityGML only in a very generic way, is built into CityJSON v1.1.1: an explicitly defined set of six metadata properties based on ISO19115. In order to enforce consistency in the use of metadata in CityJSON, only the root CityJSON object may have metadata, whereas in CityGML, gml:metaDataProperty can appear almost anywhere and contain anything. An extended set of 3D-specific metadata properties are defined by the Metadata extension [Labetski 2018].

Extensions to the core model can be created easily and quickly, and these do not require software used in the downstream to be modified, that is files containing Extensions can be processed as “normal” CityJSON files.

**Advantages of CityJSON**

The aim of CityJSON is to offer an alternative to the GML encoding of CityGML, which can be verbose and therefore complex to work with). CityJSON aims at being easy-to-use, both for reading datasets, and for creating them. It was designed with programmers in mind, so that tools and APIs supporting it can be quickly built. It was also designed to be compact (it typically compresses publicly available CityGML files by 6x).

A CityJSON file is on average about a factor 6 more compact than its CityGML equivalent, see <https://github.com/tudelft3d/cityjson/wiki/Compression-factor-for-a-few-open-CityGML-datasets> for some real-world examples.

For a factual explanation of JSON’s success on the Web, see also section 1.2 of [Masó 2017].

**Applications and use cases**

All the applications and use cases that are theoretically possible with CityGML also apply to CityJSON. A list of these applications and use cases is found on the website: <https://www.cityjson.org/applications/>.

Notice that since JSON is used for the encoding, the CityJSON developers believe that CityJSON actually expands the possibilities of CityGML since it is easier for developers to implement CityJSON in their software, especially if that software is web-based. As an example, the CityJSON developers are not aware of any JavaScript code to parse CityGML files (the complete standard), while for CityJSON it would be simpler as JSON can be deemed as “native for the web”.

**Extensions to the core model**

It is possible to easily define Extensions to the core model. CityJSON uses JSON Schemas to document and validate the data model, schemas should be seen as basically validating the syntax of a JSON document.

A CityJSON Extension is a JSON file that allows documentation of how the core data model of CityJSON may be extended, and to validate CityJSON files containing new objects and/or attributes. This is conceptually akin to the Application Domain Extensions (ADEs) in CityGML; see Section 10.13 of the official CityGML documentation. Although CityJSON Extensions have the same function as CityGML ADEs, they do not follow the same rules and thus cannot be considered as a direct JSON translation.

**Software**

Several software packages and libraries offer CityJSON support. The full list is at <https://www.cityjson.org/software/>.

It should be noticed that several programming languages have been used in implementations so far: Python, Java, C++, JavaScript, Ruby, Objective-C and Swift. This helps ensure that CityJSON is “implementable” for most developers, and not just within one environment.

**History of its development**

CityJSON was started, and is maintained, by the 3D geoinformation group at TU Delft (<https://3d.bk.tudelft.nl>). Others have since then joined its development, especially virtualcitySYSTEMS and Claus Nagel (<https://www.virtualcitysystems.de>).

The CityJSON developers invite anyone to contribute to the development and improvement of CityJSON, all discussions, issues, and developments are open to everyone on the GitHub repository of CityJSON: https://github.com/cityjson/specs

# Relationship to other OGC standards

*State whether this proposed Community standard has any dependencies on OGC standards or is itself normatively referenced by an OGC standard and list those standards, as applicable.*

CityJSON version 1.0 is already an OGC Community Standard and it is an encoding for a subset of the OGC CityGML data model (version 2.0.0).The proposed update of the CityJSON standard will support the OGC CityGML data model version 3.0. The few features that are not currently supported by the new version are either because they are seldom used, or because they would overcomplicate and decrease the strengths of the JSON encoding; see https://www.cityjson.org/conformance/v30/.

There are also capabilities for bidirectional conversion between CityJSON and CityGML. Thus, using CityJSON means that you are using the CityGML data model.

# Alignment with OGC Standards Baseline

*Describe where this proposed standard fits with respect to the existing OGC standards baseline and standards in development in the OGC and whether this proposed standard may compete with or enhance an existing OGC standard*.

CityJSON is an encoding for a subset of the OGC CityGML data model (version 3.0). It offers an alternative encoding to the official OGC GML encoding of CityGML.

CityJSON v1.1.1 is aligned with the current CityGML version (v3.0) only. Since CityGML separates the conceptual model and the encodings (such as GML and JSON), CityJSON can be considered as an encoding for CityGML v3.For future potential versions of CityJSON and CityGML there will need to be agreements in order to avoid divergence between the two standards.

# Evidence of implementation

The following implementations use the proposed Community standard (CityJSON v1.1).

* An overview of the software is at <https://www.cityjson.org/software/>
* Some scripts and example software for reading/writing/processing CityJSON in different languages are available at https://github.com/cityjson

**Implementation name**: 3dcitydb

**Date of most recent version**: 2022-05-23

**Implementation description**: The 3D City Database is a free 3D geo database to store, represent, and manage virtual 3D city models on top of a standard spatial relational database. The database model contains semantically rich, hierarchically structured, multi-scale urban objects facilitating complex GIS modeling and analysis tasks, far beyond visualization. In 2012, the 3D City Database received the Oracle Spatial Excellence Award for Education and Research. The schema of the 3D City Database is based on the OGC City Geography Markup Language (CityGML), an international standard for representing and exchanging virtual 3D city models issued by the Open Geospatial Consortium (OGC). It fully supports to read and write CityJSON.

**Implementation URL**: <https://github.com/3dcitydb/3dcitydb/>

**Is implementation complete**? Yes

**Implementation name**: citygml4j

**Date of most recent version**: 2019-08-11

**Implementation description**: citygml4j is an open source Java class library and API for facilitating work with the [OGC City Geography Markup Language (CityGML)](http://www.opengeospatial.org/standards/citygml). citygml4j makes it easy to read, process and write CityGML datasets, and to develop CityGML-aware software applications. Starting from version 2.6.0, citygml4j supports parsing and writing [CityJSON](http://www.cityjson.org/), a format for encoding a subset of the CityGML data model using JSON instead of GML. Written in Java.

**Implementation URL**: <https://github.com/citygml4j/citygml4j>

**Is implementation complete**? Yes

**Implementation name**: cjio

**Date of most recent version**: 2021-12-15

**Implementation description**: Python CLI to process and manipulate [CityJSON](http://www.cityjson.org) files. The different operators can be chained to perform several processing operations in one step, the CityJSON model goes through them and different versions of the CityJSON model can be saved as files along the pipeline. Written in Python.

**Implementation URL**: https://github.com/cityjson/cjio

**Is implementation complete**? Yes (all features of CityJSON are supported)

**Implementation name**: QGIS plugin

**Date of most recent version**: 2022-03-23

**Implementation description**: Python plugin for QGIS 3 which adds support for loading [CityJSON](http://www.cityjson.org) datasets in QGIS. Written in Python.

**Implementation URL**: https://github.com/cityjson/cityjson-qgis-plugin

**Is implementation complete**? No.

**If not, what portions of the proposed Community standard are implemented?**

Textures and CityJSONFeature are not supported.

Writing of files not supported yet, only reading/importing.

**Implementation name**: WFS implementation by VCS

**Date of most recent version**: 2019-05-13

**Implementation description**: virtualcitySYSTEMS has a WFS interface for the 3D City Database that supports CityJSON as output format. For their 3D web map client, they just added the possibility to load and visualize CityJSON data directly in the browser, either by uploading files or by querying the database via the WFS.

**Implementation URL**: not an open implementation

**Is implementation complete**? Yes.

**Implementation name**: cjval

**Date of most recent version**: 2022-02-02

**Implementation description**: A validator for CityJSON files, which validates against the CityJSON schemas. Implemented in the Rust programming language. Companion libraries are also implemented in Python (<https://github.com/cityjson/cjvalpy> ) and WebAssembly (<https://github.com/cityjson/cjval_wasm> ). The validator can run in a web-browser and does not require to upload the validated file, see <https://validator.cityjson.org/>

**Implementation URL**: <https://github.com/cityjson/cjval>

**Is implementation complete**? Yes.

**Implementation name**: RESTful access demo

**Date of most recent version**: 2021-12-24

**Implementation description**: A demo implementation of the streaming capabilities of CityJSON v1.1. See the demo in operation: <http://cityjson.pythonanywhere.com/>

**Implementation URL**: https://github.com/cityjson/restful\_demo

**Is implementation complete**? No.

**If not, what portions of the proposed Community standard are implemented?**

The demo only implements CityJSONFeature-s to test their viability.

**Implementation name**: NINJa

**Date of most recent version**: 2022-04-052019-11-21

**Implementation description**: Web-viewer and online editor for CityJSON files. Files can be visualised, modified (their attributes and feature types) and then saved. The viewer is publicly available at <https://viewer.cityjson.org>

**Implementation URL**: https://github.com/cityjson/ninja

**Is implementation complete**? No.

**If not, what portions of the proposed Community standard are implemented?**

Textures and CityJSONFeature are missing. But we are working on adding them.

*Repeat for each implementation.*

*Optionally, provide a narrative description of the extent of implementation of the proposed Community standard for those proposed standards that are very widely used.*

# Public availability

Is the proposed Community standard currently publicly available? **Yes**

URL: <https://cityjson.org>

# Supporting member(s)

*List the supporting organizations. There must be at least three OGC organizations of which at least one must be an OGC Voting Member.*

1. Geonovum
2. Delft University of Technology
3. Kadaster International
4. virtualcitySYSTEMS
5. National University of Singapore
6. [Forum Virium Helsinki Oy](https://forumvirium.fi/)
7. Ordnance Survey

# Intellectual property rights

Will the contributor retain intellectual property rights? **Yes**

If yes, the contributor will be required to work with OGC staff to properly attribute the submitter’s intellectual property rights.

If no, the contributor will assign intellectual property rights to the OGC.

NOTE: The IP is in the public domain (CC0 license), so it will not be assigned to OGC.

# 9. References

* [Ledoux 2019] LEDOUX, Hugo, ARROYO OHORI, Ken, KUMAR, Kavisha, DUKAI, Balázs, LABETSKI, Anna and VITALIS, Stelios. CityJSON: a compact and easy-to-use encoding of the CityGML data model. Open Geospatial Data, Software and Standards [online]. 17 June 2019. Vol. 4, no. 1, p. 4. ISSN 2363-7501. Available from:<https://doi.org/10.1186/s40965-019-0064-0>
* [Masó 2017] MASÓ, Joan and ZABALA, Alaitz, eds. OGC 16-122r1, Testbed-12 JSON and GeoJSON User Guide [online]. OGC User Guide. Open Geospatial Consortium, 21 June 2017. Available from:<http://docs.opengeospatial.org/guides/16-122r1.html>
* [Labetski 2018] LABETSKI, Anna, KUMAR, Kavisha, LEDOUX, Hugo and STOTER, Jantien. A metadata ADE for CityGML. Open Geospatial Data, Software and Standards, 30 November 2018. Vol.3, no. .1, p 16. Available from: <https://dx.doi.org/10.1186/s40965-018-0057-4>

# A1. Response to comments

For transparency, we include all comments that were received during the public comment period for CityJSON as a community standard work item, and our responses to these comments.