# OGC Testbed-14 Next Generation APIs

***Complex Feature Handling Engineering Report***

# Table of Contents

Publication Date: 2019-03-06

Approval Date: 2018-12-13

Submission Date: 2018-11-19

Reference number of this document: OGC 18-021

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D040

Category: Public Engineering Report

Editor: Clemens Portele

Title: OGC Testbed-14 Next Generation APIs: Complex Feature Handling Engineering Report

---

**OGC Engineering Report**

**COPYRIGHT**



**WARNING**

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

**LICENSE AGREEMENT**

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

# Chapter 1. Summary

OGC Web Feature Service (WFS) 3.0 [https://github.com/opengeospatial/WFS_FES] is a revision of the WFS standard [http://www.opengeospatial.org/standards/wfs] that proposes a modernized service architecture, that follows the current Web architecture, has a focus on the developer experience, supports the OpenAPI specification [https://www.openapis.org/], and modularizes WFS into building blocks for fine-grained access to spatial data that can be used by an Application Programming Interface (API) for data.

This document reviews the work that proposes a next generation of OGC web services ("NextGen services" or "Next Generation APIs") from the perspective of supporting complex three-dimensional (3D) data or complex data schemas. The goal is to identify the best service solution for these particular needs, whether the results are WFS 3.0 extensions or other approaches. In this context the approach of the NextGen services is not of monolithic web services, but Web API building blocks. This is an important point. The same API should be able to support requirements that currently require separate OGC web services, e.g. a WFS and a 3D Portrayal Service (3DPS).

The purpose of this work is not to preempt other next-generation discussions taking place in OGC but rather to inform and complement that work.

The report includes proposals on how to extend the NextGen service architecture with API building blocks for complex data, complex queries and 3D portrayal. WFS 3.0, Part 1, is used as the starting point for the NextGen service architecture. The proposals are based on existing requirements and use cases as well as existing support for developers to simplify implementation.

The work has found no general issues with migrating current WFS, 3DPS, Web Map Tile Service (WMTS) and Web Map Service (WMS) capabilities to the NextGen architecture. On the contrary, the NextGen approach improves the consistency of the interface and removes redundancies (e.g., between the feature access in WFS and the feature info requests in the other standards).

## 1.1. Requirements & Research Motivation

The current work on WFS 3.0 has focused on simple features and interaction, interrogation and extraction of these. However, there are more complex use cases that require 3D data and/or more complex data schemas to achieve a user-desired outcome.

This report documents real-world use cases and requirements that feature servers will have to support in the future, besides the essential capabilities specified in Part 1 of the WFS 3.0 series. These will include the following:

- Querying / filtering features based on properties of related or nested objects or structured data types, including cases where more than one level of relationship/nesting is used;
- Querying / filtering features based on expressions built from complex predicates consisting of predicate groups and combinations of logical operators;
- Querying feature data and returning only parts of the features (selected properties, a single property only, etc.);
- Access to and query of solid geometries and other geometries in a 3D Coordinate Reference

System (CRS);

- Use of responses for display in a web browser;
- Accessing different versions (including historic representations) of features.

The use cases cover aspects like CityGML, CityGML Application Domain Extensions (ADEs), 3DPS, and WFS 2.0 interactions, but use other complex data schemas as background, too.

Based on the identified requirements, this report addresses the following aspects:

- Assessment of what use cases, especially those that utilize 3D or more complex data schemas, would require more complex interactions than the current WFS 3.0, Part 1, draft would provide;
- Assessment of current OGC standards and community specifications on the basis of both supporting the uses cases captured and alignment to NextGen service approaches, including a review around the interaction between 3DPS and WFS 2.0;
- Recommendations for the most appropriate way to support these use cases within the NextGen service architecture.

Developer requirements have to be taken into account in the design. That is, the implementation of the proposed design should be as simple as possible (given the advanced, complex requirements). This includes research for available libraries that support implementations. The OGC community is not the only one implementing rich queries on complex data.

## 1.2. Prior-After Comparison

The current work on WFS 3.0 in the Web Feature Service/Filter Encoding Specification (WFS/FES) Standards Working Group (SWG) has focused on simple features and interaction, interrogation and extraction of these. However, there are more complex use cases that require 3D data and/or more complex data schemas to achieve the results that a user is looking for.

This work proposes extensions to the NextGen service architecture to support such use cases.

Initial discussions about related topics are:

- Placeholder issue in the WFS 3.0 repository for richer query capabilities: WFS 3.0 Search extension [https://github.com/opengeospatial/WFS_FES/issues/79]
- Representational State Transfer (REST) binding in the WFS 2.x and Filter Encoding Standard (FES) 2.x draft specifications: Document folder in the OGC portal (only accessible to OGC WFS/FES SWG members) [https://portal.opengeospatial.org/index.php?m=projects&a=view&project_id=390&tab=2&artifact_id=56200]

Like the work in the WFS/FES SWG on WFS 3.0, the work presented by this report has been open to the public from the beginning in order to allow early feedback from developers already during the testbed.

The formal review of the draft report before it will be submitted to the OGC Technical Committee for consideration and publication as an OGC Engineering Report will be by the WFS/FES SWG.

## 1.3. Recommendations for Future Work

See the chapter "Recommendations". The OGC Innovation Program would be ideal to continue work on the recommendations 1 to 4 based on recommendation 6 (validate and refine recommendations through implementation):

- Recommendation 1a: Specify WFS 3.0 extensions for improved fetching of feature data
- Recommendation 1b: Make CQL an OGC standard on its own
- Recommendation 2: Migrate additional OGC web service standards to the NextGen architecture
- Recommendation 3: Investigate if/how GraphQL could be used to query a dataset
- Recommendation 4: Consider the use of CityJSON for 3D city models

Recommendation 5 (develop guidance for implementing additional search capabilities) could also be a candidate for the development of a Guide as part of a Testbed.

Recommendation 7 (register media types for encodings to be used in Web APIs) is future work for the organisations that govern the i3s, 3D Tiles and CityJSON specifications.

The remainder of this section structures these recommendations in terms of potential future tasks, components, and engineering reports of future OGC Innovation Program initiatives.

The topic of GraphQL for feature data could be a candidate for a dedicated code sprint / hackathon.

### 1.3.1. Recommended Future Tasks

**Next Generation APIs**

OGC Web Feature Service (WFS) 3.0 [https://github.com/opengeospatial/WFS_FES] is a revision of the WFS standard [http://www.opengeospatial.org/standards/wfs] that proposes a modernized service architecture, that follows the current Web architecture, has a focus on the developer experience, supports the OpenAPI specification [https://www.openapis.org/], and modularizes WFS into building blocks for fine-grained access to spatial data that can be used in APIs for data.

OGC Testbed-14, the OGC Vector Tiles Pilot, the ongoing discussions in the OGC Architecture Board, the OGC Architecture DWG and the OWS Common SWG as well as various other activities outside of the OGC have advanced the architecture for a Next Generation of OGC standards for Web APIs significantly. The work in future OGC Innovation Program initiatives can build on these results to mature preliminary results and prepare them for standardization and to explore additional capabilties, resource types and encodings.

In particular, the following sub-tasks are recommended:

- Specify WFS 3.0 extensions for improved fetching of feature data
- Migrate additional resource types used in current OGC web service standards to a Next Generation architecture
- Investigate if/how GraphQL could be used to query a feature dataset
- Consider the use of CityJSON as an additional WFS 3.0 encoding for 3D city models

- Develop guidance for implementing advanced search capabilities in WFS 3.0 APIs

In addition, if work on portrayal is planned, it should also be investigated how styling should be supported in a Next Generation architecture.

### 1.3.2. Recommended Future Deliverables

#### Recommended Future Components

The following components are suggested to be deployed to test and demonstrate "complex feature handling" capabilities in Web APIs. Validation and refinement through implementation is fundamental for standards related to Web API building blocks. All requirements should be validated in multiple implementations before considering them for standardisation.

- Next Generation API server(s) with support for CQL Core predicates
- Next Generation API server(s) with support for STAC JSON queries
- Next Generation API server(s) with support for CQL Extensions predicates
- Next Generation API server(s) with support for GraphQL
- Next Generation API server(s) with support for 2D maps
- Next Generation API server(s) with support for 3D scenes and views
- Next Generation API client(s)

As usual, the client(s) should support all tested capabilities.

#### Recommended Future Engineering Reports (ER)

These Engineering Reports would document the results of the component development and be written so that the result can be used in the OGC Standards Program as initial drafts for new standards. Exceptions are the Guide, which is about guidance, not conformance and potentially the GraphQL ER, which is more about experiments and documenting the results.

- CQL Core standard (Draft) ER (see recommendation 1b)
- CQL Extensions standard (Draft) ER (see item 2 in recommendation 1a and recommendation 1b)
- Web APIs: WFS 3.0 Extensions ER (see items 1, 3 and 4 in recommendation 1a and recommendation 4)
- Web APIs: Additional Resource Types ER (see recommendation 2)
- Web APIs: GraphQL ER (see recommendation 3)
- Web APIs: Rich Queries Guide (see recommendation 5)

## 1.4. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

#### Contacts

| Name | Organization |
| --- | --- |
| Clemens Portele *(editor)* | interactive instruments GmbH |
| Volker Coors | Hochschule für Technik Stuttgart |

# 1.5. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

# Chapter 2. References

The following normative documents are referenced in this document.

- OGC: OGC 17-069, OGC Web Feature Service 3.0: Part 1 - Core, Version 3.0.0-draft.1 [https://cdn.rawgit.com/opengeospatial/WFS_FES/3.0.0-draft.1/docs/17-069.html]
- OGC: OGC 12-019, OGC City Geography Markup Language (CityGML) Encoding Standard 2.0 [https://portal.opengeospatial.org/files/?artifact_id=47842]
- OGC: OGC 15-001r4, OGC 3D Portrayal Service 1.0 [https://docs.opengeospatial.org/is/15-001r4/15-001r4.html]
- OpenAPI Initiative: OpenAPI Specification 3.0.1 [https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md]

# Chapter 3. Terms and definitions

The following terms and definitions apply:

- dataset

```
collection of data, published or curated by a single agent, and available for
access or download in one or more formats [W3C Data Catalog Vocabulary (DCAT)]
```

**NOTE** The use of 'collection' in the definition from DCAT is broader than the use of the term in WFS 3.0. See the definition of 'feature collection'.

- distribution

```
represents an accessible form of a dataset [W3C Data Catalog Vocabulary (DCAT)]
```

**NOTE** Examples are: a downloadable file, an RSS feed or a web service that provides the data.

- feature

```
abstraction of real world phenomena [ISO 19101-1:2014]
```

**NOTE** If you are unfamiliar with the term 'feature', the explanations in the W3C/OGC Spatial Data on the Web Best Practice document [1] may help, in particular the section on Spatial Things, Features and Geometry [https://www.w3.org/TR/sdw-bp/#spatial-things-features-and-geometry].

- feature collection | collection

```
a set of features from a dataset [OGC Web Feature Service 3.0: Part 1 - Core]
```

**NOTE** In WFS 3.0, 'collection' is used as a synonym for 'feature collection'. This is done to make, for example, URI path expressions shorter and easier to understand for those that are not geo-experts.

- OpenAPI definition | OpenAPI document

```
a document (or set of documents) that defines or describes an API and conforms to
the OpenAPI Specification [derived from the OpenAPI Specification]
```

## 3.1. Abbreviated terms

- COM Component Object Model
- 2D two-dimensional
- 3D three-dimensional
- 3DPS 3D Portrayal Service
- ADE Application Domain Extension
- API Application Programming Interface
- CQL Common Query Language
- CRS Coordinate Reference System
- DCAT Data Catalog Vocabulary
- DWG Domain Working Group
- FES Filter Encoding Specification
- GML Geography Markup Language
- HTML Hypertext Markup Language
- IANA Internet Assigned Numbers Authority
- JRC Joint Research Centre of the European Commission
- JSON Java Script Object Notation
- LoD Level of Detail
- NAS Normbasierte Austauschschnittstelle (Standards-based Data Exchange Interface)
- OGC Open Geospatial Consortium
- REST Representational State Transfer
- SLD Styled Layer Descriptor
- SE Symbology Encoding
- SWG Standards Working Group
- STAC SpatioTemporal Asset Catalog
- UTC Coordinated Universal Time
- W3C World Wide Web Consortium
- WFS Web Feature Service
- WMS Web Map Service
- WMTS Web Map Tile Service
- XML Extensible Markup Language

# Chapter 4. Overview

Section 5 documents real-world use cases and requirements that more advanced feature servers will have to support in the future, besides the essential capabilities specified in Part 1 of the WFS 3.0 series.

Section 6 analyses the use cases and identifies the requirements that would require more complex interactions than those supported by the current draft of WFS 3.0, Part 1.

Section 7 assesses current OGC standards and community standards with respect to the use cases and the approach to a next generation of OGC services ("NextGen services") based on the approach taken by WFS 3.0.

Section 8 provides recommendations for the most appropriate way to support the use cases within the NextGen service architecture.

# Chapter 5. Use cases

## 5.1. Introduction

This section will document real-world use cases and requirements that more advanced feature servers should support in the future, besides the essential capabilities specified in Part 1 of the WFS 3.0 series.

All query examples are used in practice, either as queries submitted by a client, in stored queries of a WFS or in Styled Layer Descriptor (SLD)/Symbology Encoding (SE) documents defining layers based on feature data.

The use cases cover aspects like CityGML, CityGML Application Domain Extensions (ADEs), 3DPS, and WFS 2.0 interactions, but use other complex data schemas, too.

We identify for each use case example why the example has been included in this report. There is at least one example for each of the following aspects:

- Querying features based on properties of related or nested objects or structured data types (one level)
- Querying features based on properties of related or nested objects or structured data types (several levels)
- Querying features based on expressions built from complex predicates consisting of predicate groups and combinations of logical operators
- Querying feature data and returning only parts of the features (selected properties, a single property only, etc.)
- Querying feature data and returning additional features linked to the selected features
- Access to and query of solid geometries and other geometries in a 3D CRS
- Use of responses for display in a web browser
- Accessing different versions (including historic representations) of features

## 5.2. Managing and using a Cadastral dataset

### 5.2.1. Description

The dataset used in this section is about cadastral parcels and related information. It uses the AFIS-ALKIS-ATKIS application schema [http://www.adv-online.de/AAA-Modell/] of the Surveying Authorities in Germany [http://www.adv-online.de/]. The GML application schema is the so-called "Standards-based Data Exchange Interface" (NAS) [http://repository.gdi-de.org/schemas/adv/nas/6.0/aaa.xsd].

The application schema is optimized for maintaining the data by the Surveying Authorities and reflects the legal requirements. As a result, the application schema contains many relationships between feature types as well as structured data types. The application schema includes information related to portrayal of the data at map scales that are traditionally used for base maps, too. For example, information is included about text placement in specific maps.

Since cadastral law permits arcs in a boundary of a cadastral parcel, Simple Feature geometries are not sufficient for storing the authoritative feature data.

Outside of the Surveying Authorities the data is often simplified / flattened to simplify the use of the data, but a number of workflows require the full use of the complex data structures included in the dataset.

As the application schema is used in Germany, it uses German terms. For better readability we show a simplified schema using English translation for each query described in this section.

### 5.2.2. Selection of protected sites

This is a relatively simple use case where features are selected based on properties of related features, i.e. features that are the value of an association role property.

The following aspects are covered by this use case:

- Querying features based on properties of related or nested objects or structured data types (one level)
- Querying feature data and returning additional features linked to the selected features

#### Data

The queries use the following feature types:

**ProtectedSite_Water (AX_SchutzgebietNachWasserrecht)**

A protected site based on laws pertaining to water and waterways. It consists of one or more zones.

**ProtectedZone (AX_Schutzzone)**

An area in a protected site with a homogeneous classification.

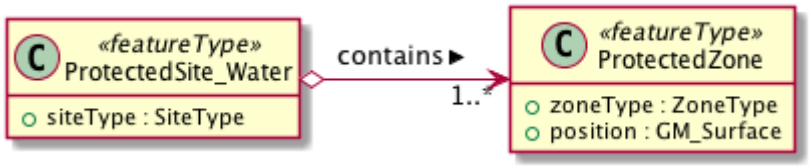


*Figure 1. UML class diagram for features in use case "Selection of protected sites"*

#### Query 1: Select just the protected site features

The following WFS query selects all ProtectedSite_Water (AX_SchutzgebietNachWasserrecht) features that have a protected zone in a given bounding box using a value reference `contains/ProtectedZone/position` (`adv:bestehtAus/adv:AX_Schutzzone/adv:position`).

```
https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?
  service=WFS&
  version=2.0.0&
  request=GetFeature&
  namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&
  typenames=adv:AX_SchutzgebietNachWasserrecht&
  filter=
  <fes:Filter
    xmlns:adv="http://www.adv-online.de/namespaces/adv/gid/6.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:Intersects>
      <fes:ValueReference>
        adv:bestehtAus/adv:AX_Schutzzone/adv:position
      </fes:ValueReference>
      <gml:Envelope
        srsName="http://www.opengis.net/def/crs/epsg/0/25832">
        <gml:lowerCorner>360000 5600000</gml:lowerCorner>
        <gml:upperCorner>370000 5700000</gml:upperCorner>
      </gml:Envelope>
    </fes:Intersects>
  </fes:Filter>
```

The result is a feature collection with four ProtectedSite_Water (AX_SchutzgebietNachWasserrecht) features.

Live link to invoke the query on a WFS… [https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?service=WFS&version=2.0.0&request=GetFeature&namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&typenames=adv:AX_SchutzgebietNachWasserrecht&filter=%3Cfes%3AFilter%20xmlns%3D%22http%3A%2F%2Fwww.adv-online.de%2Fnamespaces%2Fadv%2Fgid%2F6.0%22%20xmlns%3Agml%3D%22http%3A%2F%2Fwww.opengis.net%2Fgml%2F3.2%22%20xmlns%3Afes%3D%22http%3A%2F%2Fwww.opengis.net%2Ffes%2F2.0%22%3E%0A%20%20%3Cfes%3AIntersects%3E%0A%20%20%20%20%3Cfes%3AValueReference%3Eadv%3AbestehtAus%2Fadv%3AAX_Schutzzone%2Fadv%3Aposition%3C%2Ffes%3AValueReference%3E%0A%20%20%20%20%3Cgml%3AEnvelope%20srsName%3D%22http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2Fepsg%2F0%2F25832%22%3E%0A%20%20%20%20%3Cgml%3AlowerCorner%3E360000%205600000%3C%2Fgml%3AlowerCorner%3E%0A%20%20%20%20%3Cgml%3AupperCorner%3E370000%205700000%3C%2Fgml%3AupperCorner%3E%0A%20%20%20%20%3C%2Fgml%3AEnvelope%3E%0A%20%20%3C%2Ffes%3AIntersects%3E%0A%3C%2Ffes%3AFilter%3E]

### Query 2: Select also the related protected zone features

The following WFS query selects not only the ProtectedSite_Water (AX_SchutzgebietNachWasserrecht) features that have a protected zone in a given bounding box, but uses the WFS `resolve` and `resolvedepth` parameters to retrieve all the zone features in the same query.

```
https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?
  service=WFS&
  version=2.0.0&
  request=GetFeature&
  namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&
  typenames=adv:AX_SchutzgebietNachWasserrecht&
  resolve=local&
  resolvedepth=1&
  filter=
  <fes:Filter
    xmlns:adv="http://www.adv-online.de/namespaces/adv/gid/6.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:Intersects>
      <fes:ValueReference>
        adv:bestehtAus/adv:AX_Schutzzone/adv:position
      </fes:ValueReference>
      <gml:Envelope
        srsName="http://www.opengis.net/def/crs/epsg/0/25832">
        <gml:lowerCorner>360000 5600000</gml:lowerCorner>
        <gml:upperCorner>370000 5700000</gml:upperCorner>
      </gml:Envelope>
    </fes:Intersects>
  </fes:Filter>
```

The result is a feature collection with four ProtectedSite_Water (AX_SchutzgebietNachWasserrecht) features and one ProtectedZone (AX_Schutzzone) feature in each site.

Live link to invoke the query on a WFS… [https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?service=WFS&version=2.0.0&request=GetFeature&namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&typenames=adv:AX_SchutzgebietNachWasserrecht&resolve=local&resolvedepth=1&filter=%3Cfes%3AFilter%20xmlns%3D%22http%3A%2F%2Fwww.adv-online.de%2Fnamespaces%2Fadv%2Fgid%2F6.0%22%20xmlns%3Agml%3D%22http%3A%2F%2Fwww.opengis.net%2Fgml%2F3.2%22%20xmlns%3Afes%3D%22http%3A%2F%2Fwww.opengis.net%2Ffes%2F2.0%22%3E%0A%20%20%3Cfes%3AIntersects%3E%0A%20%20%20%20%3Cfes%3AValueReference%3Eadv%3AbestehtAus%2Fadv%3AAX_Schutzzone%2Fadv%3Aposition%3C%2Ffes%3AValueReference%3E%0A%20%20%20%20%3Cgml%3AEnvelope%20srsName%3D%22http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2Fepsg%2F0%2F25832%22%3E%0A%20%20%20%20%3Cgml%3AlowerCorner%3E360000%205600000%3C%2Fgml%3AlowerCorner%3E%0A%20%20%20%20%3Cgml%3AupperCorner%3E370000%205700000%3C%2Fgml%3AupperCorner%3E%0A%20%20%20%20%3C%2Fgml%3AEnvelope%3E%0A%20%20%3C%2Ffes%3AIntersects%3E%0A%3C%2Ffes%3AFilter%3E]

### 5.2.3. Select the owners of cadastral parcels in an area

This use case is similar to the previous one, but uses value references along multiple associations.

The following aspects are covered by this use case:

- Querying features based on properties of related or nested objects or structured data types (several levels)

### Data

The dataset is the same as in the previous use case.

The example query uses the following feature types. This is simplified, the actual schema and data is much more complex and reflects the legal requirements of the German land register.

**CadastralParcel (AX_Flurstueck)**

A cadastral parcel.

**Record (multiple feature types)**

An entry in the land register.

**Person (AX_Person)**

A person that has some rights or responsibilities related to one or more parcels.

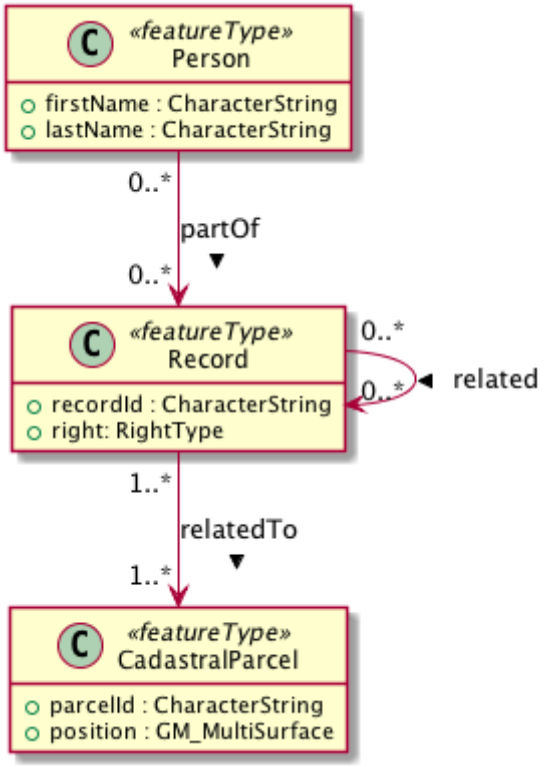


*Figure 2. UML class diagram for features in use case "Select the owners of cadastral parcels in an area"*

### Query

The following WFS query selects all Person (AX_Person) features, that are related to cadastral parcels in a bounding box, e.g. own the parcel or have some rights. The filter uses a value reference along multiple associations: `partOf/Record/relatedTo/CadastralParcel/position` (the first value reference) or `partOf/Record/related/Record/relatedTo/CadastralParcel/position` (the second value reference).

```
https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?
  service=WFS&
  version=2.0.0&
  request=GetFeature&
  namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&
  typenames=adv:AX_Person&
  filter=
  <fes:Filter
    xmlns:adv="http://www.adv-online.de/namespaces/adv/gid/6.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:Or>
      <fes:Intersects>
        <fes:ValueReference>
          adv:weistAuf/adv:AX_Namensnummer/adv:istBestandteilVon/
          adv:AX_Buchungsblatt/adv:bestehtAus/adv:AX_Buchungsstelle/
          adv:grundstueckBestehtAus/adv:AX_Flurstueck/adv:position
        </fes:ValueReference>
        <gml:Envelope
          srsName="http://www.opengis.net/def/crs/epsg/0/25832">
          <gml:lowerCorner>361000 5610000</gml:lowerCorner>
          <gml:upperCorner>362000 5620000</gml:upperCorner>
        </gml:Envelope>
      </fes:Intersects>
      <fes:Intersects>
        <fes:ValueReference>
          adv:weistAuf/adv:AX_Namensnummer/adv:istBestandteilVon/
          adv:AX_Buchungsblatt/adv:bestehtAus/adv:AX_Buchungsstelle/
          adv:an/adv:AX_Buchungsstelle/adv:grundstueckBestehtAus/
          adv:AX_Flurstueck/adv:position
        </fes:ValueReference>
        <gml:Envelope
          srsName="http://www.opengis.net/def/crs/epsg/0/25832">
          <gml:lowerCorner>361000 5610000</gml:lowerCorner>
          <gml:upperCorner>362000 5620000</gml:upperCorner>
        </gml:Envelope>
      </fes:Intersects>
    </fes:Or>
  </fes:Filter>
```

The result is a feature collection with the person features matching the query.

Due to privacy regulations, the land register data is not open data and no live query link can be provided.

### 5.2.4. Select versions of cadastral parcels based on their temporal validity

Often, the history of a dataset is important. The example that we are using here is a cadastral parcel dataset, where it can be important to know the state of the parcels at a point in the past.

There are two options for how this is typically handled in application schemas.

One approach is that the features are in fact feature versions. That is, different versions of the same feature / real-world entity are each represented as separate features. This is the approach we are considering in this use case. To avoid confusion we use the terms "version" and "real-world entity" in the description of this use case instead of "feature" which could mean the feature or a specific version of the feature.

The advantage of this approach is that no specific temporal support is required in clients processing the data. This pattern is therefore frequently used with data that is used in map-based GIS clients, for example, with datasets provided by mapping or cadastral agencies.

The other approach is to model the feature properties as timestamped sequences of values. GML supports this approach with the Dynamic Features pattern. The downside of this approach is that clients and servers must support this specific pattern, which typically requires customized software. A domain that is using this approach is the aviation domain.

The following aspects are covered by this use case:

- Accessing different versions (including historic representations) of features

**Data**

The dataset is the same as in the first use case.

As described above, the features in the application schema are versions of a real-world entity, valid for a given time period.

All versions of the same real-world entity have the same `gml:identifier`. If multiple versions occur in the same GML document, a timestamp will be added to the `gml:id` attribute, otherwise the identifier of the real-world entity will be used.

Each version has information about the lifespan of the version at hand. i.e., each version has a timestamp when this version has been added to the dataset. If the version is still valid, there is no timestamp for the end of the version validity. If the version (or the real-world entity) is no longer valid in the dataset, a timestamp for the end is added.

Each timestamp is given in Coordinated Universal Time (UTC), the granularity is seconds.

If a new version is added due to a change in a property, the new version will have a start timestamp that is one second after the end timestamp of the previous version.

The example query uses the following feature type. The actual schema and data is more complex and has been simplified to the relevant aspects for this use case.

**CadastralParcel (AX_Flurstueck)**

A cadastral parcel.

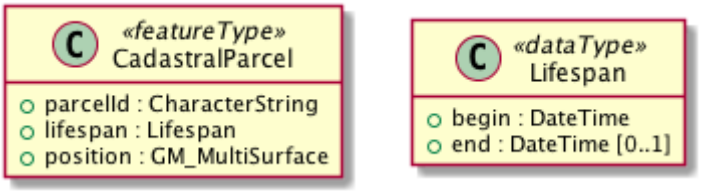


*Figure 3. UML class diagram for features in use case "Select versions of cadastral parcels based on their temporal validity"*

### Query

The following WFS query selects all CadastralParcel (AX_Flurstueck) versions that have been inserted into the dataset on July 1st, 2017.

```
https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?
  service=WFS&
  version=2.0.0&
  request=GetFeature&
  namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&
  typenames=adv:AX_Flurstueck&
  filter=
  <fes:Filter
    xmlns:adv="http://www.adv-online.de/namespaces/adv/gid/6.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:During>
      <fes:ValueReference>
        adv:lebenszeitintervall/adv:AA_Lebenszeitintervall/adv:beginnt
      </fes:ValueReference>
      <gml:TimePeriod gml:id="TP1">
      <gml:begin>
        <gml:TimeInstant gml:id="TI1">
          <gml:timePosition>2017-07-01T00:00:00Z</gml:timePosition>
        </gml:TimeInstant>
      </gml:begin>
      <gml:end>
        <gml:TimeInstant gml:id="TI2">
          <gml:timePosition>2017-07-01T23:59:59Z</gml:timePosition>
        </gml:TimeInstant>
      </gml:end>
      </gml:TimePeriod>
    </fes:During>
  </fes:Filter>
```

The result is a feature collection with eight CadastralParcel (AX_Flurstueck) features.

Live link to invoke the query on a WFS… [https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?service=WFS&version=2.0.0&request=GetFeature&namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&typenames=adv:AX_Flurstueck&filter=%3Cfes%3AFilter%0A%20%20%20%20xmlns%3Aadv%3D%22http%3A%2F%2Fwww.adv-online.de%2Fnamespaces%2Fadv%2Fgid%2F6.0%22%0A%20%20%20%20xmlns%3Agml%3D%22http%3A%2F%2Fw

ww.opengis.net%2Fgml%2F3.2%22%0A%20%20%20%20xmlns%3Afes%3D%22http%3A%2F%2Fwww.opengis.net%2Ffes%2F2.0%22%3E%0A%20%20%20%20%3Cfes%3ADuring%3E%0A%20%20%20%20%20%20%3Cfes%3AValueReference%3Eadv%3Alebenszeitintervall%2Fadv%3AAAA_Lebenszeitintervall%2Fadv%3Abeginnt%3C%2Ffes%3AValueReference%3E%0A%20%20%20%20%20%20%3Cgml%3ATimePeriod%20gml%3Aid%3D%22TP1%22%3E%0A%20%20%20%20%20%20%3Cgml%3Abegin%3E%0A%20%20%20%20%20%20%20%20%3Cgml%3ATimeInstant%20gml%3Aid%3D%22TI1%22%3E%0A%20%20%20%20%20%20%20%20%20%20%3Cgml%3AtimePosition%3E2017-07-01T00%3A00%3A00Z%3C%2Fgml%3AtimePosition%3E%0A%20%20%20%20%20%20%20%20%3C%2Fgml%3ATimeInstant%3E%0A%20%20%20%20%20%20%3C%2Fgml%3Abegin%3E%0A%20%20%20%20%20%20%3Cgml%3Aend%3E%0A%20%20%20%20%20%20%20%20%3Cgml%3ATimeInstant%20gml%3Aid%3D%22TI2%22%3E%0A%20%20%20%20%20%20%20%20%20%20%3Cgml%3AtimePosition%3E2017-07-01T23%3A59%3A59Z%3C%2Fgml%3AtimePosition%3E%0A%20%20%20%20%20%20%20%20%3C%2Fgml%3ATimeInstant%3E%0A%20%20%20%20%20%20%3C%2Fgml%3Aend%3E%0A%20%20%20%20%20%20%3C%2Fgml%3ATimePeriod%3E%0A%20%20%20%20%3C%2Ffes%3ADuring%3E%0A%20%20%3C%2Ffes%3AFilter%3E]

Note that the dataset accessible via the WFS only includes valid versions, because WFS 2.0 does not include a simple mechanism to handle versions in queries and most users, especially those using a map-based GIS client, would be surprised to receive multiple features from the WFS representing the same real-world entity. All of those versions would be drawn on a map at the same time.

There is an opportunity with WFS 3.0 to support datasets with versions natively.

See also the related discussion in the W3C/OGC Spatial Data on the Web Best Practice document [https://www.w3.org/TR/sdw-bp/#bp-dataversioning].

### 5.2.5. Select cadastral parcels for rendering with a specific style

A common requirement is to present features in a dataset on a map (or in a 3D scene). In this use case we look at rendering feature data on a 2D map, for display in a web browser.

This may be implemented using a WFS 2.0 as the backend, i.e. the rendering engine is a WFS client and then renders the data, either directly in the browser or in a server, for example, a WMS 1.3.

For server-side rendering, the data will typically be rendered closer to the database and not via a WFS 2.0 interface - for performance reasons. For client-side rendering, the data will typically not use GML, but a format that is optimized for the rendering purpose. Nevertheless, the use case is still relevant in the context of complex feature handling, for at least two reasons:

- Style information in the OGC standards baseline uses Symbology Encoding and the feature selection mechanisms are the same as in WFS 2.0 - both use the Filter Encoding standard.
- In this report, we are not limited to WFS only, but we want to consider other aspects that are relevant for spatial Web APIs, too. As NextGen services will have to be able to support API building blocks for providing maps, scenes, tiles, etc., the related query aspects need to be considered, too.

The following aspects are covered by this use case:

- Querying features based on expressions built from complex predicates consisting of predicate groups and combinations of logical operators
- Use of responses for display in a web browser

### Data

The dataset is the same as in the first use case.

The example query uses the following feature types. The actual schema and data are more complex and have been simplified to the relevant aspects for this use case.

**CadastralParcel (AX_Flurstueck)**

A cadastral parcel.

**Text (AP_PTO)**

A map text for display on a map for a feature.

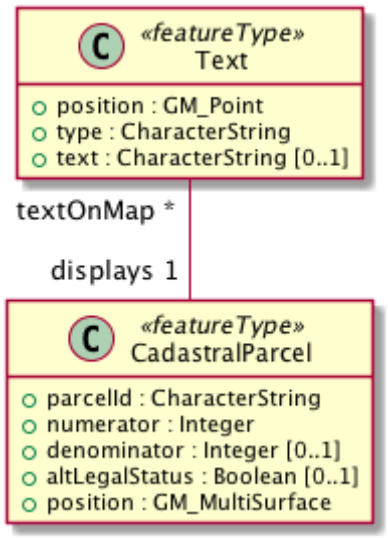


*Figure 4. UML class diagram for features in use case "Select cadastral parcels for rendering with a specific style"*

### Query

Rich, standardized symbology rule sets exist for the cadastral datasets consisting of a large number of selection rules and feature styles.

We will use rules RUL06410 and RUL06420 from the ALKIS portrayal catalogue [http://sg.geodatenzentrum.de/web_public/adv/sk/alkis/docAlkisFB/html/SYCALFB1xLAY00001xRST00001ById.html] as an example. The rules select all cadastral parcels that meet the following criteria (for display of the parcel number on the map):

- Parcels in a local district are identified using a numerator ("Zähler") and an optional denominator ("Nenner"). The example rules only apply to parcels with a denominator. The value reference is `numerator` (`adv:flurstuecksnummer/adv:AX_Flurstuecksnummer/adv:nenner`).
- In addition, all of the following conditions must be met:
  - Another organization other than the land register may be legally responsible for some parcels. This is indicated in a boolean attribute for an alternative legal status ("abweichenderRechtszustand"). The example rules only apply to parcels for which the attribute is either missing or `false`. The value reference is `altLegalStatus` (`adv:abweichenderRechtszustand`).

- The application schema includes special feature types to capture map placement information. A typical example is a Text object (AP_PTO), which may be used to provide a fixed location for a text on the map ("position") or to provide a different text ("schriftinhalt") than the default text derived from the properties of the real-world thing. An association exists between the cadastral parcel and the Text objects that contain information overriding the default portrayal on the map ("inversZu_dientZurDarstellungVon_AP_PTO"). Since a map may contain multiple texts for a feature, there is also a type property ("art") to distinguish different text types. The example rules only apply to parcels that have an associated Text object for displaying the parcel number on the map (type is "ZAE_NEN"). The value reference is `textOnMap/Text[type = 'ZAE_NEN']` (`adv:inversZu_dientZurDarstellungVon_AP_PTO/adv:AP_PTO[adv:art = 'ZAE_NEN']`).

The difference between the two rules RUL06410 and RUL06420 is whether the text on the map is taken from the `numerator` attribute of the cadastral parcel feature or from the `text` attribute of the associated Text object.

The following WFS query selects all CadastralParcel (AX_Flurstueck) features that are rendered using the example portrayal rules. The `<fes:Filter>` part would be the same in a portrayal rule according to the Symbology Encoding standard as used in a WMS/SLD.

```
https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?
  service=WFS&
  version=2.0.0&
  request=GetFeature&
  namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&
  typenames=adv:AX_Flurstueck&
  filter=
  <fes:Filter xmlns:adv="http://www.adv-online.de/namespaces/adv/gid/6.0"
  xmlns:gml="http://www.opengis.net/gml/3.2"
  xmlns:fes="http://www.opengis.net/fes/2.0">
  <fes:And>
    <fes:Not>
      <fes:PropertyIsNull>
        <fes:ValueReference>
          adv:flurstuecksnummer/adv:AX_Flurstuecksnummer/adv:nenner
        </fes:ValueReference>
      </fes:PropertyIsNull>
    </fes:Not>
    <fes:And>
      <fes:Or>
        <fes:PropertyIsNull>
          <fes:ValueReference>
            adv:abweichenderRechtszustand
          </fes:ValueReference>
        </fes:PropertyIsNull>
        <fes:PropertyIsEqualTo>
          <fes:ValueReference>
            adv:abweichenderRechtszustand
          </fes:ValueReference>
          <fes:Literal>false</fes:Literal>
        </fes:PropertyIsEqualTo>
      </fes:Or>
      <fes:Not>
        <fes:PropertyIsNull>
          <fes:ValueReference>
            adv:inversZu_dientZurDarstellungVon_AP_PTO/adv:AP_PTO[adv:art = 'ZAE_NEN']
          </fes:ValueReference>
        </fes:PropertyIsNull>
      </fes:Not>
    </fes:And>
  </fes:And>
  </fes:Filter>
```

The result is a feature collection with more than 234,000 CadastralParcel (AX_Flurstueck) features.

Live link to invoke the query on a WFS (with resultType=hits due to the large number of features)…
[https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?service=WFS&version=2.0.0&
request=GetFeature&namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&
typenames=adv:AX_Flurstueck&filter=%3Cfes%3AFilter%20xmlns%3Aadv%3D%22http%3A%2F%2Fwww.adv-
online.de%2Fnamespaces%2Fadv%2Fgid%2F6.0%22%0A%20%20xmlns%3Agml%3D%22http%3A%2F%2Fwww.open

gis.net%2Fgml%2F3.2%22%0A%20%20xmlns%3Afes%3D%22http%3A%2F%2Fwww.opengis.net%2Ffes%2F2.0%22%3E%0A%20%20%3Cfes%3AAnd%3E%0A%20%20%20%20%3Cfes%3ANot%3E%0A%20%20%20%20%20%20%3Cfes%3APropertyIsNull%3E%0A%20%20%20%20%20%20%20%20%3Cfes%3AValueReference%3E%0A%20%20%20%20%20%20%20%20%20%20adv%3Aflurstuecksnummer%2Fadv%3AAX_Flurstuecksnummer%2Fadv%3Anenner%0A%20%20%20%20%20%20%20%20%20%20%3C%2Ffes%3AValueReference%3E%0A%20%20%20%20%20%20%3C%2Ffes%3APropertyIsNull%3E%0A%20%20%20%20%3C%2Ffes%3ANot%3E%0A%20%20%20%20%3Cfes%3AAnd%3E%0A%20%20%20%20%20%20%3Cfes%3AOr%3E%0A%20%20%20%20%20%20%20%20%3Cfes%3APropertyIsNull%3E%0A%20%20%20%20%20%20%20%20%20%20%3Cfes%3AValueReference%3E%0A%20%20%20%20%20%20%20%20%20%20adv%3AabweichenderRechtszustand%0A%20%20%20%20%20%20%20%20%20%20%3C%2Ffes%3AValueReference%3E%0A%20%20%20%20%20%20%20%20%3C%2Ffes%3APropertyIsNull%3E%0A%20%20%20%20%20%20%20%20%3Cfes%3APropertyIsEqualTo%3E%0A%20%20%20%20%20%20%20%20%20%20%3Cfes%3AValueReference%3E%0A%20%20%20%20%20%20%20%20%20%20adv%3AabweichenderRechtszustand%0A%20%20%20%20%20%20%20%20%20%20%3C%2Ffes%3AValueReference%3E%0A%20%20%20%20%20%20%20%20%20%20%3Cfes%3ALiteral%3Efalse%3C%2Ffes%3ALiteral%3E%0A%20%20%20%20%20%20%20%20%3C%2Ffes%3APropertyIsEqualTo%3E%0A%20%20%20%20%20%20%3C%2Ffes%3AOr%3E%0A%20%20%20%20%20%20%3Cfes%3ANot%3E%0A%20%20%20%20%20%20%20%20%3Cfes%3APropertyIsNull%3E%0A%20%20%20%20%20%20%20%20%20%20%3Cfes%3AValueReference%3E%0A%20%20%20%20%20%20%20%20%20%20adv%3AinversZu_dientZurDarstellungVon_AP_PTO%2Fadv%3AAP_PTO%5Badv%3Aart%20%3D%20%27ZAE_NEN%27%5D%0A%20%20%20%20%20%20%20%20%20%20%3C%2Ffes%3AValueReference%3E%0A%20%20%20%20%20%20%20%20%3C%2Ffes%3APropertyIsNull%3E%0A%20%20%20%20%20%20%3C%2Ffes%3ANot%3E%0A%20%20%20%20%3C%2Ffes%3AAnd%3E%0A%20%20%3C%2Ffes%3AAnd%3E%0A%20%20%3C%2Ffes%3AFilter%3E&resultType=hits]

## 5.2.6. Selection of building parts of a building

This use case has been added to include a query that - while not overly complex - cannot be executed with most WFS implementations as it would require support for spatial joins.

Including this use case does not imply that WFS 3.0 should include an extension that supports such a query. However, as spatial relations are an important aspect of spatial data, we should at least include it in our considerations, even if we recommend to include explicit spatial relations in the feature representations, consistent with the recommendations of the W3C/OGC Spatial Data on the Web Best Practice document [https://www.w3.org/TR/sdw-bp/#bp-linking-2].

**Data**

The AFIS-ALKIS-ATKIS application schema distinguishes between buildings and building parts, where a building part is a part of a building with different characteristics, for example, a different number of floors. The two-dimensional (2D) footprint geometry of a building part is within the footprint geometry of the building.

| NOTE | The CityGML Building and BuildingPart features were originally modelled after the cadastral model in Germany. |
|---|---|

As the AFIS-ALKIS-ATKIS application schema is designed for maintaining the cadastral datasets, they do not contain a (redundant) association to identify the building parts within a build (as the relationship can be determined from the footprint geometries).

The queries use the following feature types:

**Building (AX_Gebaeude)**

A permanent construction that must be recorded due its significance for the cadastre.

**BuildingPart (AX_Bauteil)**

A part of a Building with distinct or special characteristics.


*Figure 5. UML class diagram for features in use case "Selection of building parts of a building"*

**Query**

Two subsequent queries are required.

The first query retrieves the building using its identifier (`DENW45AL0000lxrJ`) in order to determine the footprint geometry of the building.

```
https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?
  service=WFS&
  version=2.0.0&
  request=GetFeature&
  resourceId=DENW45AL0000lxrJ
```

Live link to invoke the query on a WFS... [https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?service=WFS&version=2.0.0&request=GetFeature&resourceId=DENW45AL0000lxrJ]

The geometry can now be used to retrieve all building parts of that building.

```
https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?
  service=WFS&
  version=2.0.0&
  request=GetFeature&
  namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&
  typenames=adv:AX_Bauteil&
  filter=
  <fes:Filter
    xmlns:adv="http://www.adv-online.de/namespaces/adv/gid/6.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:Intersects>
      <fes:ValueReference>
        adv:position
      </fes:ValueReference>
      <gml:Polygon gml:id="o31001.id.29956334.position.Geom_0"
srsName="urn:ogc:def:crs:EPSG::25832">
        <gml:exterior>
          <gml:LinearRing>
            <gml:posList>
              377034.58 5658143.873 377036.274 5658136.338 377036.438 5658135.613
              377038.889 5658136.168 377039.429 5658136.291 377043.42 5658137.194
              377043.262 5658137.895 377041.564 5658145.455 377041.193 5658145.371
              377038.311 5658144.715 377034.58 5658143.873
            </gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:Polygon>
    </fes:Intersects>
  </fes:Filter>
```

The result is a feature collection with two BuildingPart (AX_Bauteil) features.

Live link to invoke the query on a WFS… [https://www.wfs.nrw.de/geobasis/wfs_nw_alkis_aaa-modell-basiert?service=WFS&version=2.0.0&request=GetFeature&namespaces=xmlns(adv,http://www.adv-online.de/namespaces/adv/gid/6.0)&typenames=adv:AX_Bauteil&filter=%3Cfes%3AFilter%0A%20%20%20%20xmlns%3Aadv%3D%22http%3A%2F%2Fwww.adv-online.de%2Fnamespaces%2Fadv%2Fgid%2F6.0%22%0A%20%20%20%20xmlns%3Agml%3D%22http%3A%2F%2Fwww.opengis.net%2Fgml%2F3.2%22%0A%20%20%20%20xmlns%3Afes%3D%22http%3A%2F%2Fwww.opengis.net%2Ffes%2F2.0%22%3E%0A%20%20%20%20%3Cfes%3AIntersects%3E%0A%20%20%20%20%20%20%3Cfes%3AValueReference%3Eadv%3Aposition%3C%2Ffes%3AValueReference%3E%0A%3Cgml%3APolygon%20gml%3Aid%3D%22o31001.id.29956334.position.Geom_0%22%20srsName%3D%22urn%3Aogc%3Adef%3Acrs%3AEPSG%3A%3A25832%22%20srsDimension%3D%222%22%3E%0A%3Cgml%3Aexterior%3E%0A%3Cgml%3ALinearRing%3E%0A%3Cgml%3AposList%3E377034.58%205658143.873%20377036.274%205658136.338%20377036.438%205658135.613%20377038.889%205658136.168%20377039.429%205658136.291%20377043.42%205658137.194%20377043.262%205658137.895%20377041.564%205658145.455%20377041.193%205658145.371%20377038.311%205658144.715%20377034.58%205658143.873%3C%2Fgml%3AposList%3E%0A%3C%2Fgml%3ALinearRing%3E%0A%3C%2Fgml%3Aexterior%3E%0A%3C%2Fgml%3APolygon%3E%0A%20%20%20%20%3C%2Ffes%3AIntersects%3E%0A%20%20%3C%2Ffes%3AFilter%3E]

# 5.3. Building heating demand simulation and visualization

## 5.3.1. Description

This section is about using 3D city models to simulate a building's heating demand and visualize the results in a web-based 3D scene. Why is this relevant? The building sector has large potential for energy efficiency gains and $CO_2$-reductions and is thus a priority area for achieving the ambitious climate and energy targets for 2020 and 2050 in the European Union [2]. In order to reach the 2% energy refurbishment rate promoted by the European Union and to realize long-term climate neutral communities, information about the current and future demand is necessary in order to develop strategies for policy making as well as to raise awareness of the citizens. Some pioneering work has already been done worldwide [3], [4], [5], [6]. As an input for the simulation, a building model in CityGML is usually used. The use case is split into several parts:

- open a website to display a 3D map using 3D Portrayal Service and select a polygonal area / district
- query 3D building geometry inside the polygonal area from a CityGML data store using WFS 3.0 as an input to the simulation
- using 3D Portrayal Service and WFS 3.0 to query the simulation result and visualize it in a 3D scene by building Id, by address, and by polygonal area
- extend the use case to the INSPIRE 3D building module in addition to CityGML

The following aspects are covered:

- Querying features based on properties of related or nested objects or structured data types (one level / several levels)
- Querying feature data and returning only parts of the features (selected properties, a single property only, etc.)
- Querying feature data and returning additional features linked to the selected features
- Access to and query of solid geometries and other geometries in a 3D CRS
- Use of responses for display in a web browser

## 5.3.2. Data

- Open data 3D city model of New York as used in Testbed-13 S3D performance. CityGML LoD 1/2 Building Model, no Textures http://www1.nyc.gov/site/doitt/initiatives/3d-building.page; heating demand simulation is available (monthly energy balance) for Manhattan as "as is" simulation (simulated with SimStadt Software, HFT Stuttgart).
- Open data 3D model of Essen as used in the Energy and Location Pilot of the the Joint Research Centre (JRC) of the European Commission. CityGML LoD 1/2 Building Model, no Textures; heating demand simulation is available (monthly energy balance) as "as is" simulation, medium refurbishment scenario, advanced refurbishment scenario (simulated with SimStadt Software, HFT Stuttgart).

- Open data 3D model of Essen, a test area converted from CityGML to INSPIRE building model using HALE Studio.

### 5.3.3. Open a webpage to display a 3D map using 3D Portrayal Service and select a polygonal area / district

This has been showcased and implemented in Testbed-13 [7]. The implementation is available and could be further used and extended in Testbed-14 with the New York data set. The experiment evaluated the complete flow of data from its originating CityGML format to a web-enabled visualization with Cesium via OGC's 3D Portrayal Service (3DPS). This data flow included the conversion from the CityGML data format served by GeoRocket, to 3D Tiles dataset, and the import of the 3D Tiles dataset to the 3DPS Framework.

A public demonstration is available at: http://tb13.igd.fraunhofer.de:8080/Apps/Sandcastle/index.html?src=3DPS_r.html&label=Showcases.

The following aspects are covered by this use case:

- Use of responses for display in a web browser (3DPS getScene Region query)

**Query**

A 3DPS getScene request example from the New York showcase:

http://tb13.igd.fraunhofer.de:8082/3dps?SERVICE=3DPS&VERSION=1.0&REQUEST=GetScene&LAYERS=manhattan&FORMAT=text/html&CRS=EPSG:4326&BOUNDINGBOX=-74.00635826977239,40.71778771238832,-73.97393297660074,40.75070138933127

In the existing implementation, 3D Tiles is used as content delivery format. See OGC Testbed-13 – 3D Tiles and I3S Interoperability and Performance Engineering Report [7], experiment 2 [http://docs.opengeospatial.org/per/17-046.html#Experiment2] for further details.

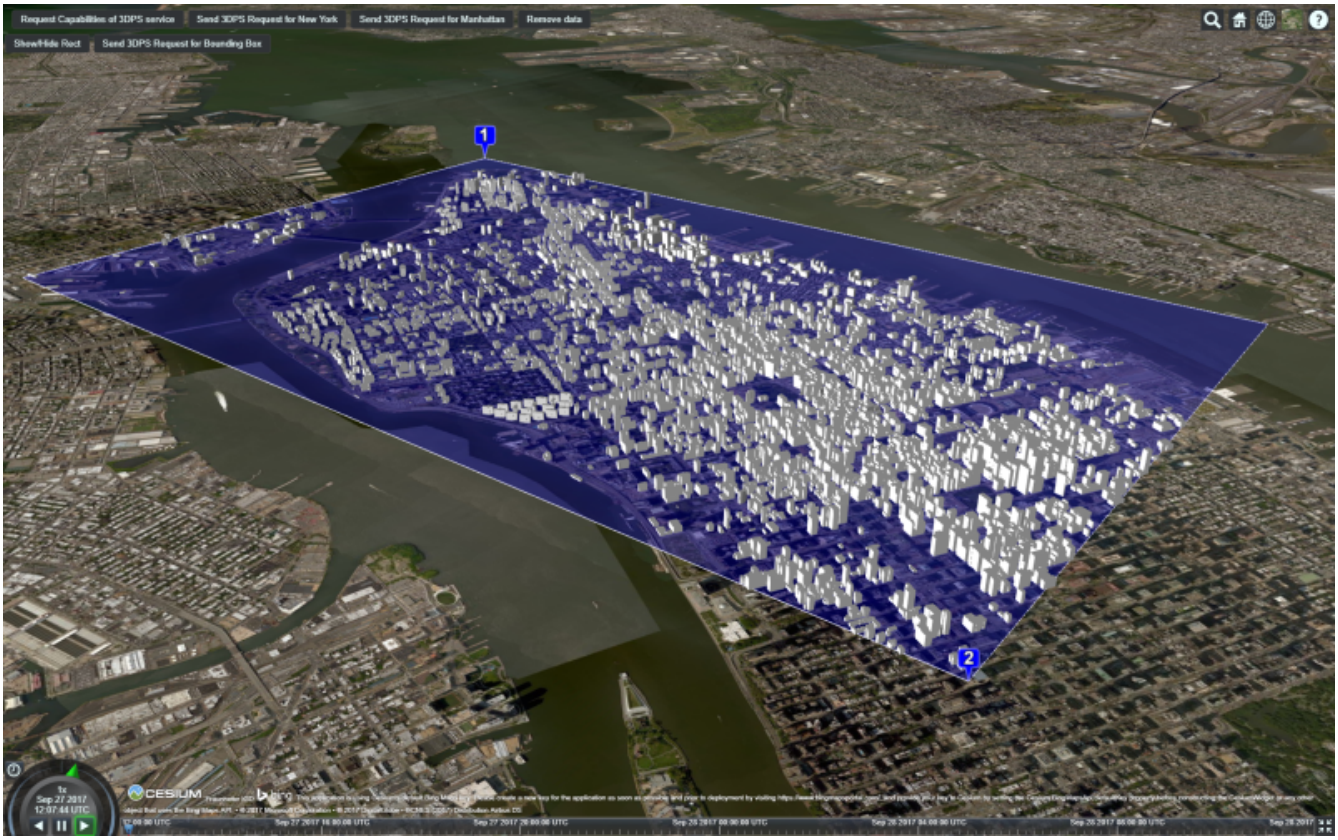

*Figure 6. 3DPS getScene request to select an (bounding box) area of the New York data set*

### 5.3.4. Using 3D Portrayal Service and WFS to query the simulation result and visualize it in a 3D scene by building Id

In Testbed-13, the 3DPS getFeatureInfo request has been used to query heating demand (simulation result) per building from a server at HFT Stuttgart. See video [https://www.youtube.com/watch?v=AbyCQKH-PC4&index=17&list=PLQsQNjNIDU87So4QmiSCKFmE0fv567s3m&t=0s] on the Testbed-13 S3D Performance Experiment #2.

It would make sense to use the building id to retrieve additional attributes from a WFS. However, the building geometry is no longer required because as it is already available.

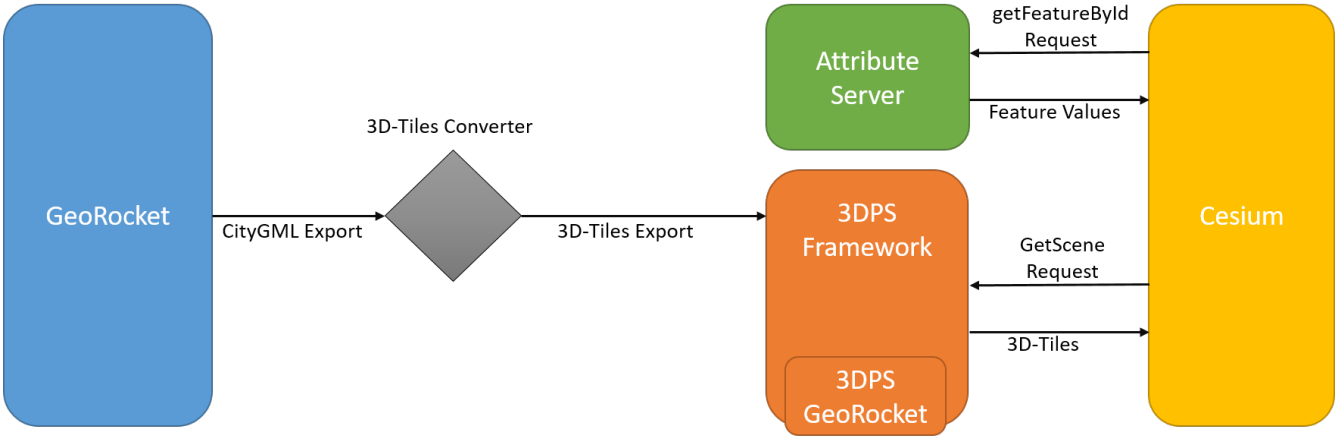


*Figure 7. The dataflow from GeoRocket to the visualized 3D Tiles, which are requested via the 3DPS queries (from Testbed-13, S3D performance)*

The following aspects are covered by this use case:

- Use of responses for display in a web browser (3DPS getScene Region query)
- Querying feature data and returning only parts of the features (selected properties, a single property only, etc.)

**Query**

A 3DPS getFeatureInfo request example from the New York showcase:

http://81.169.187.7:3100/api/v2/endpoint?service=3DPS&version=1.0&
request=GetFeatureInfoByObjectId&objectid=uuid_2824afd6-00e5-42ac-ab95-ec868595dc5a&
dataset=ny_dataset

As this is requesting feature properties, such a request could also be directed to a WFS 2.0 with the building model. The building would be selected based on its identifier using the `resourceId` parameter. The `property` parameter could be used to select only the desired properties (in the example below: the building function, the measured height and heat information); properties that are not needed, for example, the geometry, would be excluded from the response.

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(bldg,http://www.opengis.net/citygml/building/2.0)&
  typenames=bldg:Building&
  resourceId=uuid_2824afd6-00e5-42ac-ab95-ec868595dc5a&
  propertyName=function,measuredHeight,heat
```

WFS 3.0 should include an extension that supports such a capability, too.

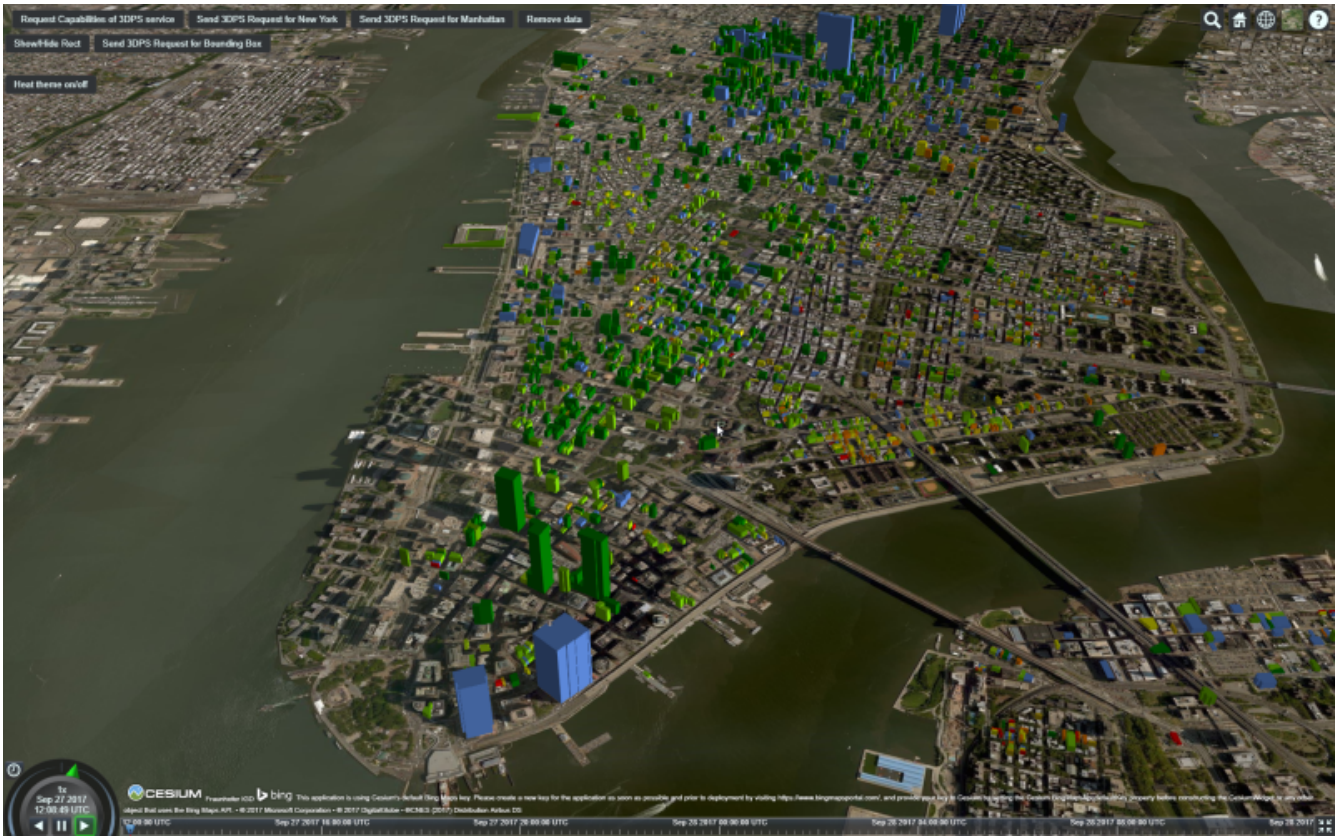


*Figure 8. Manhattan dataset with simulated heat demand, provided by HFT Stuttgart, and color coded in to the building's appearance. (from Testbed-13, S3D performance)*

### 5.3.5. Query a feature from a city model by id

Most important in this use case is that solids are supported as feature geometries.

In addition, it needs to be discussed how to deal with the Level of Detail (LoD) concept of CityGML. The CityGML model that is retrieved will typically be used by another process - in our example the simulation of the heating demand of that building.

#### Query

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(bldg,http://www.opengis.net/citygml/building/2.0)&
  typenames=bldg:Building&
  resourceId=TWINHOUSE1
```

In WFS 3.0, the building would be identified by a URI, for example, http://www.example.com/my-city-model/collections/buildings/items/TWINHOUSE1.

As mentioned above, the most important aspect in this query is that the WFS supports solid geometries and is able to return features with such geometries.

A WFS 2.0 would return the building feature in a `wfs:FeatureCollection`.

```
<wfs:FeatureCollection xmlns:xAL="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" xmlns:gml
="http://www.opengis.net/gml" xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
xmlns:wfs="http://www.opengis.net/wfs/2.0" xmlns:gen=
"http://www.opengis.net/citygml/generics/2.0" xmlns:core=
"http://www.opengis.net/citygml/2.0" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.opengis.net/citygml/building/2.0
http://schemas.opengis.net/citygml/building/2.0/building.xsd
http://www.opengis.net/wfs/2.0 http://schemas.opengis.net/wfs/2.0/wfs.xsd
http://www.opengis.net/citygml/generics/2.0
http://schemas.opengis.net/citygml/generics/2.0/generics.xsd" timeStamp="2018-03-
28T15:01:47" numberMatched="2" numberReturned="2">
 <wfs:member>
  <wfs:FeatureCollection timeStamp="2018-03-28T15:01:47" numberMatched="1"
numberReturned="1">
   <wfs:member>
    <bldg:Building gml:id="TWINHOUSE1">
     <gml:boundedBy>
      <gml:Envelope srsName="crs:EPSG::31468" srsDimension="3">
       <gml:lowerCorner>-8.0E-15 0.0 0.0</gml:lowerCorner>
       <gml:upperCorner>10.04 10.04 6.4</gml:upperCorner>
      </gml:Envelope>
     </gml:boundedBy>
     <core:creationDate>2018-03-20</core:creationDate>
     <bldg:lod1Solid>
      <gml:Solid gml:id="UUID_836b4b28-24d9-4e83-906a-98f4364d351f">
       <gml:exterior>
        <gml:CompositeSurface gml:id="UUID_2ac22267-11d4-48f0-b63d-c417228d1968">
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_e379198f-7e10-43e8-8737-851cece07579">
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_e379198f-7e10-43e8-8737-851cece07579_0_">
             <gml:posList srsDimension="3">2.0E-15 10.04 0.0 4.0E-15 10.04 1.0E-13
-0.0 0.0 0.0 2.0E-15 10.04 0.0</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_0e264d5e-3034-43fc-b65f-2b231ef5907b">
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_0e264d5e-3034-43fc-b65f-2b231ef5907b_0_">
             <gml:posList srsDimension="3">4.0E-15 10.04 1.0E-13 4.0E-15 0.0 1.0E-13
-0.0 0.0 0.0 4.0E-15 10.04 1.0E-13</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_c8dbcf60-8f0e-43f1-a1ef-ed43620dbfb1">
```

```
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_c8dbcf60-8f0e-43f1-a1ef-ed43620dbfb1_0_">
             <gml:posList srsDimension="3">4.0E-15 10.04 1.0E-13 10.04 10.04 0.0 10.04
0.0 0.0 4.0E-15 0.0 1.0E-13 4.0E-15 10.04 1.0E-13</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_22c99934-a675-4b42-97af-f73874d1aabb">
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_22c99934-a675-4b42-97af-f73874d1aabb_0_">
             <gml:posList srsDimension="3">10.04 0.0 6.4 10.04 0.0 0.0 10.04 10.04 0.0
10.04 10.04 6.4 10.04 0.0 6.4</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_13db3bd0-6210-414c-b884-3bd2099c9680">
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_13db3bd0-6210-414c-b884-3bd2099c9680_0_">
             <gml:posList srsDimension="3">10.04 10.04 6.4 10.04 10.04 0.0 4.0E-15
10.04 1.0E-13 2.0E-15 10.04 0.0 -8.0E-15 10.04 6.39999999999999 10.04 10.04
6.4</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_024dfb16-831c-4404-9c94-cdda06aaca86">
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_024dfb16-831c-4404-9c94-cdda06aaca86_0_">
             <gml:posList srsDimension="3">2.0E-15 10.04 0.0 -0.0 0.0 0.0 -8.0E-15
10.04 6.39999999999999 2.0E-15 10.04 0.0</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_a9f8e079-5033-49ed-851a-aae7f9454dd8">
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_a9f8e079-5033-49ed-851a-aae7f9454dd8_0_">
             <gml:posList srsDimension="3">-8.0E-15 10.04 6.39999999999999 -0.0 0.0
0.0 -8.0E-15 0.0 6.39999999999999 -8.0E-15 10.04 6.39999999999999</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_a6d3c8c7-ace0-4e48-b8c1-ca18cd5a814d">
```

```
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_a6d3c8c7-ace0-4e48-b8c1-ca18cd5a814d_0_">
             <gml:posList srsDimension="3">10.04 0.0 6.4 -8.0E-15 0.0 6.39999999999999
-0.0 0.0 0.0 4.0E-15 0.0 1.0E-13 10.04 0.0 0.0 10.04 0.0 6.4</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
         <gml:surfaceMember>
          <gml:Polygon gml:id="UUID_c1b51c00-2dbc-45d2-9c93-c9b396382780">
           <gml:exterior>
            <gml:LinearRing gml:id="UUID_c1b51c00-2dbc-45d2-9c93-c9b396382780_0_">
             <gml:posList srsDimension="3">-8.0E-15 10.04 6.39999999999999 -8.0E-15
0.0 6.39999999999999 10.04 0.0 6.4 10.04 10.04 6.4 -8.0E-15 10.04
6.39999999999999</gml:posList>
            </gml:LinearRing>
           </gml:exterior>
          </gml:Polygon>
         </gml:surfaceMember>
        </gml:CompositeSurface>
       </gml:exterior>
      </gml:Solid>
     </bldg:lod1Solid>
     <bldg:lod1TerrainIntersection>
      <gml:MultiCurve>
       <gml:curveMember>
        <gml:LineString>
         <gml:posList srsDimension="3">10.04 0.0 0.0 10.04 10.04 0.0</gml:posList>
        </gml:LineString>
       </gml:curveMember>
       <gml:curveMember>
        <gml:LineString>
         <gml:posList srsDimension="3">-0.0 0.0 0.0 10.04 0.0 0.0</gml:posList>
        </gml:LineString>
       </gml:curveMember>
       <gml:curveMember>
        <gml:LineString>
         <gml:posList srsDimension="3">2.0E-15 10.04 0.0 -0.0 0.0 0.0</gml:posList>
        </gml:LineString>
       </gml:curveMember>
       <gml:curveMember>
        <gml:LineString>
         <gml:posList srsDimension="3">2.0E-15 10.04 0.0 10.04 10.04 0.0</gml:posList>
        </gml:LineString>
       </gml:curveMember>
      </gml:MultiCurve>
     </bldg:lod1TerrainIntersection>
    </bldg:Building>
   </wfs:member>
  </wfs:FeatureCollection>
 </wfs:member>
```

INSPIRE recommends the use of `wfs:FeatureCollection`, too, but if the data is not accessed via a WFS other feature collections may be used as well.

CityGML itself also includes a feature collection element, `core:CityModel`, that may be used, if the service interface is not a WFS 2.0.

```
<core:CityModel xmlns:smil20="http://www.w3.org/2001/SMIL20/" xmlns:grp=
"http://www.opengis.net/citygml/cityobjectgroup/1.0" xmlns:smil20lang=
"http://www.w3.org/2001/SMIL20/Language" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:base="http://www.citygml.org/citygml/profiles/base/1.0" xmlns:luse=
"http://www.opengis.net/citygml/landuse/1.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:frn=
"http://www.opengis.net/citygml/cityfurniture/1.0" xmlns:dem=
"http://www.opengis.net/citygml/relief/1.0" xmlns:tran=
"http://www.opengis.net/citygml/transportation/1.0" xmlns:wtr=
"http://www.opengis.net/citygml/waterbody/1.0" xmlns:tex=
"http://www.opengis.net/citygml/texturedsurface/1.0" xmlns:core=
"http://www.opengis.net/citygml/1.0" xmlns:xAL=
"urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" xmlns:bldg=
"http://www.opengis.net/citygml/building/1.0" xmlns:sch=
"http://www.ascc.net/xml/schematron" xmlns:app=
"http://www.opengis.net/citygml/appearance/1.0" xmlns:veg=
"http://www.opengis.net/citygml/vegetation/1.0" xmlns:gml="http://www.opengis.net/gml"
xmlns:gen="http://www.opengis.net/citygml/generics/1.0">
  <core:cityObjectMember>
    <bldg:Building gml:id="TWINHOUSE1">
      ...
    </bldg:Building>
  </core:cityObjectMember>
</core:CityModel>
```

In WFS 3.0 the response will be determined by the encoding and the requirements of the conformance class for that encoding.

None of the WFS 3.0 Core conformance classes for encodings supports solid geometries.

However, for responses that are CityGML 2.0, the same `wfs30:FeatureCollection` element could be used that is also used in the WFS 3.0 conformance classes for the GML Simple Feature encodings [https://cdn.rawgit.com/opengeospatial/WFS_FES/3.0.0-draft.1/docs/17-069.html#_requirement_class_geography_markup_language_gml_simple_features_profile_level_0].

Another option could be a WFS 3.0 that returns CityJSON.

### 5.3.6. Select buildings in a 2D region from a city model

In this example, all buildings in a rectangular region are requested and the selected building features are returned in a feature collection.

**Query**

Here is an example of a WFS 2.0 query:

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(bldg,http://www.opengis.net/citygml/building/2.0)&
  typenames=bldg:Building&
  BBOX=-74,40.7,-73.96,40.8
```

Using an instance of GeoRocket containing the New York City CityGML model developed in Testbed-13 the request could also be:

http://192.44.35.62:63020/store/?search=-74,40.7,-73.96,40.8

This link points to a live service, but returns a large response. A more manageable response can be retrieved with a smaller bounding box [http://192.44.35.62:63020/store/?search=-74,40.75,-73.999999,40.7500001].

In WFS 3.0, the request is also supported by the query capabilities of the Core, for example:

http://www.example.com/my-city-model/collections/buildings/items?bbox=-74,40.7,-73.96,40.8

### 5.3.7. Select buildings based on nested features or properties

These examples have been provided by Claus Nagel, virtualcitySYSTEMS. They cover the following query categories:

- Querying features based on properties of related or nested objects or structured data types (several levels)
- Access to and query of solid geometries and other geometries in a 3D CRS

**Query 1: Select buildings based on their ground surface geometry**

This query retrieves all buildings having one or more ground surfaces whose LoD 2 geometry intersects with a given geometry. `bldg:GroundSurface` is a nested feature.

In this example, the query geometry is a multi-surface with 3D coordinate values.

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(bldg,http://www.opengis.net/citygml/building/2.0)&
  typenames=bldg:Building&
  filter=
  <fes:Filter
    xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:Intersects>
      <fes:ValueReference>
        bldg:boundedBy/bldg:GroundSurface/bldg:lod2MultiSurface
      </fes:ValueReference>
      <gml:MultiSurface srsName="http://www.opengis.net/def/crs/EPSG/0/?????">
        <gml:surfaceMember>
          <gml:Polygon>
            <gml:exterior>
              <gml:LinearRing>
                <gml:posList>
                  21498.400088101323 17386.16611967112 31.123
                  <!-- ... -->
                </gml:posList>
              </gml:LinearRing>
            </gml:exterior>
          </gml:Polygon>
        </gml:surfaceMember>
      </gml:MultiSurface>
    </fes:Intersects>
  </fes:Filter>
```

The result is shown as an image as the Extensible Markup Language (XML) response itself is too verbose to show, and is not open data.

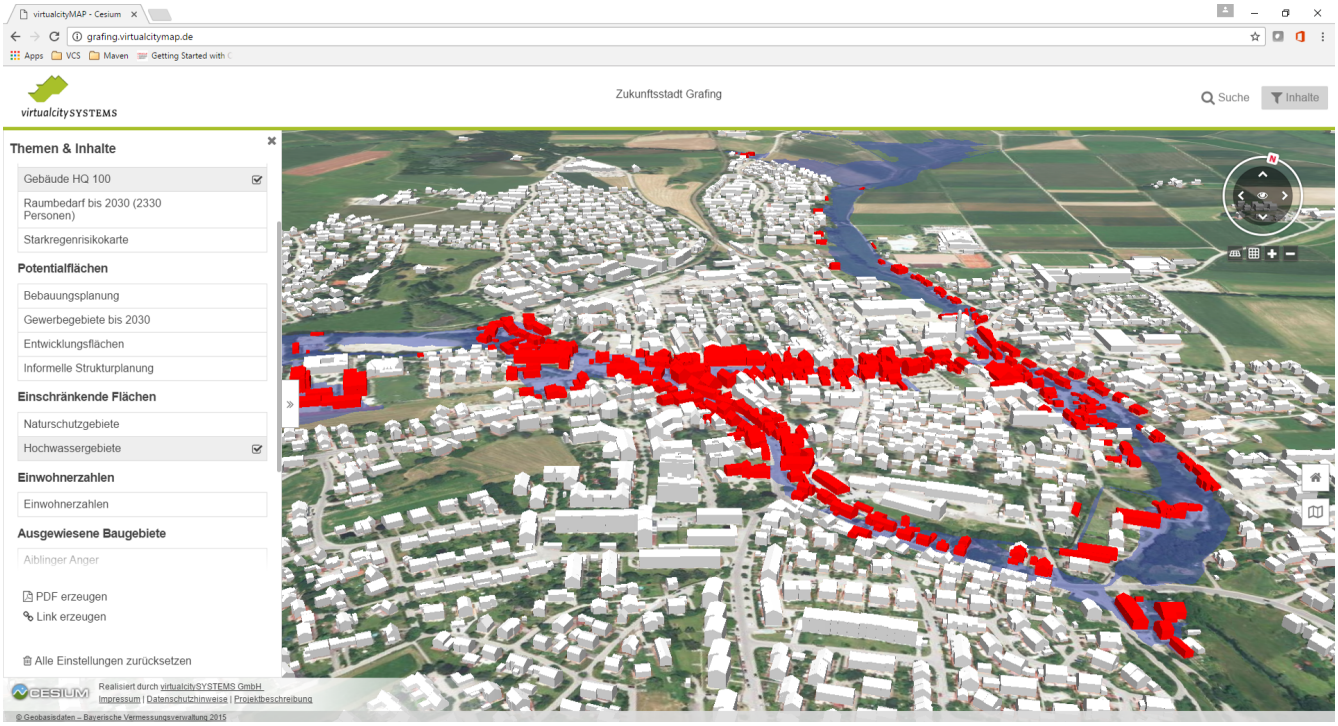


*Figure 9. Get all buildings having one or more ground surfaces whose LoD 2 geometry intersects with a given geometry (Ground Surface is a nested feature in CityGML)*

### Query 2: Select buildings along a road

This query retrieves all buildings along a given road using the road name.

`core:Address` is a nested feature, and xAL requires access to an entire subtree of XML elements.

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(bldg,http://www.opengis.net/citygml/building/2.0)&
  typenames=bldg:Building&
  filter=
  <fes:Filter
    xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
    xmlns:core="http://www.opengis.net/citygml/2.0"
    xmlns:xAL="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:PropertyIsLike wildCard="*" singleChar="." escapeChar="\">
      <fes:ValueReference>

bldg:address/core:Address/core:xalAddress/xAL:AddressDetails/xAL:Country/xAL:Locality/
xAL:Thoroughfare/xAL:ThoroughfareName
      </fes:ValueReference>
      <fes:Literal>Unter den Linden*</fes:Literal>
    </fes:PropertyIsLike>
  </fes:Filter>
```

#### Query 3: Select trees within a buffer of an implicit geometry

Get all trees that are given by an LoD 3 template geometry and where this geometry is within a distance to a given geometry. `core:ImplicitGeometry` is a complex data type.

| NOTE | In CityGML 3.0 ImplicitGeometry may become a feature type, too. |
|---|---|

In this example, the geometry is a 3D point.

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(veg,http://www.opengis.net/citygml/vegetation/2.0)&
  typenames=veg:SolitaryVegetationObject&
  filter=
  <fes:Filter
    xmlns:veg="http://www.opengis.net/citygml/vegetation/2.0"
    xmlns:core="http://www.opengis.net/citygml/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:DWithin>
      <fes:ValueReference>
        veg:lod3ImplicitRepresentation/core:ImplicitGeometry/core:relativeGMLGeometry
      </fes:ValueReference>
      <gml:Point srsName="http://www.opengis.net/def/crs/EPSG/0/?????">
        <gml:pos>21498.400088101323 17386.16611967112 145.34675</gml:pos>
      </gml:Point>
      <fes:Distance uom="m">800</fes:Distance>
    </fes:DWithin>
  </fes:Filter>
```

The result is shown as an image as the XML response itself is too verbose to show, and is not open data.

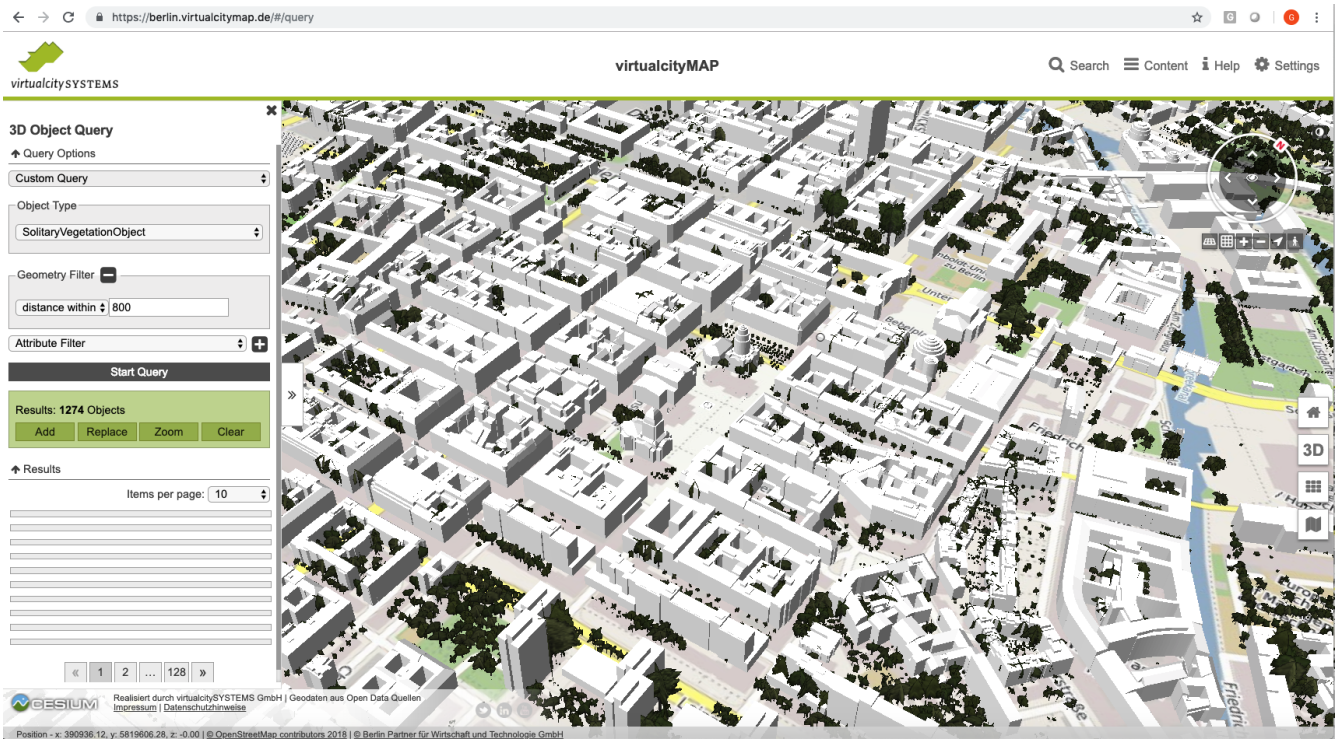


*Figure 10. get all vegetation within a given distance of a point*

### Query 4: Select buildings based on ADE information

Get all buildings that have a thermal zone which contains a thermal boundary whose `u` value is greater than a given value. This example uses the CityGML Energy ADE 1.0 extension which adds energy information to the CityGML base model.

The query involves three nested features: `energy:ThermalZone`, `energy:ThermalBoundary` and `energy:Construction`.

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(bldg,http://www.opengis.net/citygml/building/2.0)&
  typenames=bldg:Building&
  filter=
  <fes:Filter
    xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
    xmlns:energy="http://www.sig3d.org/citygml/2.0/energy/1.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
      <fes:PropertyIsGreaterThan>
        <fes:ValueReference>

energy:thermalZone/energy:ThermalZone/energy:boundedBy/energy:ThermalBoundary/energy:c
onstruction/energy:Construction/energy:uValue
        </fes:ValueReference>
        <fes:Literal>2.5</fes:Literal>
      </fes:PropertyIsGreaterThan>
    </fes:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

**Query 5: Select roads with a bicycle lane**

This query retrieves all roads with a traffic lane for bicycles.

This query involves the nested feature `tran:TrafficArea`.

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(tran,http://www.opengis.net/citygml/transportation/2.0)&
  typenames=tran:Road
  filter=
  <fes:Filter
    xmlns:tran="http://www.opengis.net/citygml/transportation/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:PropertyIsEqualTo matchCase="false">
      <fes:ValueReference>
        tran:trafficArea/tran:TrafficArea/tran:function
      </fes:ValueReference>
      <fes:Literal>cycleLane</fes:Literal>
    </fes:PropertyIsEqualTo>
  </fes:Filter>
```

**Query 6: Select buildings suitable for photovoltaic panels**

Get all buildings having one or more roof surfaces that are suitable for mounting photovoltaic panels (the attribute `pc_class` stores the suitability class which has been precomputed.

`bldg:RoofSurface` is a nested feature.

```
https://www.example.com/wfs?
  service=WFS&
  version=2.0.2&
  request=GetFeature&
  namespaces=xmlns(bldg,http://www.opengis.net/citygml/building/2.0)&
  typenames=bldg:Building&
  filter=
  <fes:Filter
    xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
    xmlns:gen="http://www.opengis.net/citygml/generics/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:fes="http://www.opengis.net/fes/2.0">
    <fes:PropertyIsBetween>
      <fes:ValueReference>

bldg:boundedBy/bldg:RoofSurface/gen:intAttribute[@gen:name='pv_class']/gen:value
      </fes:ValueReference>
      <fes:LowerBoundary>
        <fes:Literal>2</fes:Literal>
      </fes:LowerBoundary>
      <fes:UpperBoundary>
        <fes:Literal>3</fes:Literal>
      </fes:UpperBoundary>
    </fes:PropertyIsBetween>
  </fes:Filter>
```

# Chapter 6. Requirements analysis

## 6.1. General remarks

This section analyses the use cases documented in the previous chapter and identifies the requirements that would require more complex interactions than those supported by the current draft of WFS 3.0, Part 1 (Core).

Requirements that are supported by design by all implementations of WFS 3.0 Core are not discussed.

We split the analysis between requirements related to the data structures used in the data, the requirements related to querying such data and requirements for other approaches to organizing the data.

In terms of encodings, this analysis documents the support for the requirements in the feature encodings supported by WFS 3.0 Core, i.e. GeoJSON and the GML Simple Feature Profile. Requirements related to a Hypertext Markup Language (HTML) representation of features or collections of them are not discussed.

| | |
|---|---|
| **NOTE** | The identified requirements are based on the use cases documented in the previous chapter and the requirements for this study. They are not intended to exhaustively cover all potential use cases related to handling complex features. However, if there are other requirements related to complex feature handling that should be considered in this report, maybe we can still add them here, if these are relevant for the recommendations and we can reference a source for the requirement. |

## 6.2. Data structures

### 6.2.1. Structured property values

**Requirement "multiple-values"**

Support for properties with multiple values.

In an application schema, the property has a maximum multiplicity greater than "1".

WFS 3.0 Core and the GeoJSON and GML Simple Feature Profile Level 2 conformance classes already support this requirement.

GeoJSON does not place any restrictions on the contents of `properties`, i.e., the natural approach to encoding such properties in GeoJSON would be a JSON array.

GML Simple Feature Level 2 is similar as it only constrains the spatial geometries used, but has no constraints on the other feature properties. In GML, multiple values are represented by repeated property elements.

**Requirement "nested-structures"**

Support for properties with structured values.

In an application schema, the property has value type that is neither a geometry, another feature nor a literal value (like Integer or CharacterString).

Again, WFS 3.0 Core and the GeoJSON / GML Simple Feature Profile Level 2 conformance classes already support this requirement.

In GeoJSON, the natural approach is to represent such values as a JSON object.

In GML, the value of the property is an object element representing an instance of the data type.

For example, the following XML shows a GML feature with a property `property` that has two values, each a data type with two properties (`sub-property1` and `sub-property2`). The other properties are suppressed:

```
<Feature gml:id="f1">
  ...
  <property>
    <DataType>
      <sub-property1>value1</sub-property1>
      <sub-property2>value2</sub-property2>
    </DataType>
  </property>
  <property>
    <DataType>
      <sub-property1>value3</sub-property1>
      <sub-property2>value4</sub-property2>
    </DataType>
  </property>
  ...
</Feature>
```

The GeoJSON representation would be:

```
{
  "type": "Feature",
  "id": "f1",
  "geometry": { ... },
  "properties": {
    ...,
    "property": [
      {
        "sub-property1": "value1",
        "sub-property2": "value2"
      },
      {
        "sub-property1": "value3",
        "sub-property2": "value4"
      }
    ],
    ...
  }
}
```

Even though GeoJSON and GML support such complex data structures, software tools are sometimes limited to simple values only. In that case, the data structures would have to be flattened. The following GeoJSON snippet is a flattened version of the GeoJSON feature above.

```
{
  "type": "Feature",
  "id": "f1",
  "geometry": { ... },
  "properties": {
    ...,
    "property-1.sub-property1": "value1",
    "property-1.sub-property2": "value2",
    "property-2.sub-property1": "value3",
    "property-2.sub-property2": "value4",
    ...
  }
}
```

If application schemas according to ISO 19109 are used, the ShapeChange [8] can help to create flattened implementation schemas [https://shapechange.net/transformations/flattener/], i.e. where all feature properties have a literal value. ShapeChange is a Java tool that takes application schemas constructed according to ISO 19109 from a UML model and derives implementation representations.

### 6.2.2. Relationships / links

**Requirement "relationships"**

Support for properties where the value is a link to another resource, identified by a URI.

In an application schema, the property is typically a relationship role - or in some cases an application-schema-specific link data type (External references in the German cadastre application schema or in CityGML that have a URI as a value are such data types).

The GML encoding in a WFS 3.0 Core supports such properties already using Xlink references (again, only in Simple Feature Profile Level 2, not in Level 0).

GeoJSON has no concept of links, but specifies extension points and no constraints on the content of "properties". The links could thus be implemented in multiple ways:

- Inside the additional "links" JSON array that WFS 3.0 already uses to include links to alternate representations, etc.
- Inside the "properties" JSON array as string-valued properties where each relationship is a URI and the property name represents the relation type.
- Similar to the previous option, but using the link data type from WFS 3.0 Core [https://github.com/opengeospatial/WFS_FES/blob/master/core/openapi/schemas/link.yaml] as the value.

Spatial relationships between features are a somewhat special case as they can be derived by spatial analysis (see the discussion in the use cases and the W3C/OGC Spatial Data on the Web Best Practice document [https://www.w3.org/TR/sdw-bp/#bp-linking-2]), but they can use the same pattern in an encoding.

The W3C/OGC Spatial Data on the Web Working Group discussed whether the most commonly used spatial relation types should be registered with Internet Assigned Numbers Authority (IANA), but eventually deferred this [https://github.com/w3c/sdw/issues/215]. Note that schema.org now has a pending change to support them [https://pending.schema.org/GeospatialGeometry].

### 6.2.3. Geometries

WFS 3.0 supports 3D coordinates but not 3D solids. WFS 3.0 Core is therefore currently limited to 2D geometries in a 2D or 3D coordinate reference system (elevation values may be included for each position). All curves and surface boundaries are restricted to linear interpolation (plus circular arc interpolation in the GML Simple Feature conformance classes). This is not sufficient for the use cases identified in the previous chapter.

**Requirement "multiple-geometries"**

Support for multiple geometry properties per feature.

An example are geometries at different map scales / levels of details.

| | |
|---|---|
| **NOTE** | For cases where a feature has multiple spatial properties, WFS 3.0 Core specifies that it is the decision of the server whether only a single spatial geometry property is used to determine the extent or all relevant geometries. |

GML, including the Simple Feature Profiles Levels 0 and 2, supports this, but GeoJSON does not.

That is, in cases where this capability is important, another JSON encoding has to be used or the features have to be split so that there is one feature per spatial geometry property.

Another option could be to include additional geometries inside the "properties" JSON array, but GeoJSON-aware software would not identify these values as geometries. Such an option should, therefore, not be considered.

**Requirement "3D-geometries"**

Support for 3D coordinate reference systems and solid geometries.

This is fully supported by GML, but not the GML Simple Feature Profiles or GeoJSON as both encodings do not support solids. That is, none of the conformance classes in WFS 3.0 Core is able to support this requirement.

**Requirement "shared-geometries"**

Support for using the same geometry objects in the geometries of multiple features.

Again, this is supported by GML, but neither by the GML Simple Feature Profiles nor by GeoJSON.

**Requirement "non-linear-interpolation"**

Support for using arcs, splines or other interpolation of curves or surface boundaries.

Again, this is supported by GML, but neither by the GML Simple Feature Profiles (with the exception of arcs) nor by GeoJSON.

### 6.2.4. Validation against schemas

**Requirement "schema-validation"**

Support for validating features and feature collections against a schema.

Typical examples are validation against an XML Schema or a JSON Schema.

The current draft of WFS 3.0 Core includes - as a stopgap - a recommendation to include relationships to the schema documents. Another option would be `/schema` paths.

The OpenAPI Initiative is discussing more robust support ("alternative schemas") [https://github.com/OAI/OpenAPI-Specification/issues/1532] for this requirement in the OpenAPI specification. OGC should wait for the resolution of this development before looking for other solutions.

## 6.3. Queries

### 6.3.1. Query expressions

As the use cases have shown, support for richer data structures is not only required for representing the features in JSON or XML, but these data structures have to be accessible in queries, too.

**Requirement "query-properties-with-multiple-values"**

The capability to specify whether a predicate on a feature property must be met by all values for a feature or by at least one.

In FES 2.0 this is supported by the matchAction parameter [http://docs.opengeospatial.org/is/09-026r2/09-

026r2.html#50].

**Requirement "query-nested-properties"**

The capability to specify predicates on properties of a nested data type or a related feature. The "nesting" may be multiple levels.

In WFS 2.0 and FES 2.0 this is supported by the use of XPath expressions [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html#37] in value references, including for cases that require the traversal of a link to a related feature.

If the data includes explicit spatial relationships, these may be used for filtering, too.

**Requirement "query-logical-operators"**

The capability to specify query expressions built from complex predicates consisting of predicate groups and combinations of logical operators.

This requires support for grouping as well as the standard logical operators [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html#68] specified in FES 2.0.

### 6.3.2. Query using spatial geometries

WFS 3.0 Core only requires support for spatial queries using 2D bounding boxes [https://cdn.rawgit.com/opengeospatial/WFS_FES/3.0.0-draft.1/docs/17-069.html#_parameter_bbox].

**Requirement "query-2D-geometries"**

The capability to filter features based on their relationship to a point, curve or surface geometry in a 2D coordinate reference system.

`Intersects` is the most important relation, but in general the full set of standard operators [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html#59] should be supported.

**Requirement "query-buffers"**

The capability to filter based on a geometry created from an existing geometry by buffering it.

`DWithin` and `Beyond` are the standard operators [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html#59] in FES 2.0.

**Requirement "query-3D-geometries"**

The capability to filter features in 3D coordinate reference systems and using solid geometries.

### 6.3.3. Querying different versions

Using the `time` parameter, WFS 3.0 Core already supports access to features at a certain moment or period in time, but currently the default is always "no temporal filter".

For datasets where the features are versions of a real-world entity, valid for a given time period, the default response would return all versions. It would be more helpful, to change the default behavior for such datasets.

**Requirement "query-current-data"**

The capability to select by default only features that are valid "now" (for datasets where the features are versions of a real-world entity, valid for a given time period).

This has two aspects: being able to specify a different default value for `time` ("now" instead of "no filter") and to be able to explicitly state values for the indefinite past and future. The latter issue is already under discussion [https://github.com/opengeospatial/WFS_FES/issues/155] by the WFS 3.0 community.

If a time period is used in `time` the response for such datasets could include multiple versions of the same real-world entity.

This is a requirement that would need to be supported in the Core.

### 6.3.4. Optimizing the query response

Some of the use cases benefit from post-processing the feature or feature collection that has been selected.

**Requirement "embed-related-features"**

Support for including related features in the representation of a feature.

In some use cases this will avoid repeated, additional requests to the server to access the related features as their information is needed by the client application.

The related feature may either be embedded inside the property representing the relationship or - in particular where that relationship is always represented by a hyperlink - elsewhere in the document, referenced by a local link.

WFS 2.0 supports this capability using a set of resolve parameters [http://docs.opengeospatial.org/is/09-025r2/09-025r2.html#53].

**Requirement "return-subset"**

Support for returning only a subset of a feature properties.

This could be any combination, for example (links go to existing discussion in the WFS 3.0 development):

- just the id [https://github.com/opengeospatial/WFS_FES/issues/13],
- without the geometry [https://github.com/opengeospatial/WFS_FES/issues/16],
- or any subset of the feature properties, including a single property.

WFS 2.0 supports this capability using so-called projection clauses [http://docs.opengeospatial.org/is/09-025r2/09-025r2.html#96] as well as using the GetPropertyValue operation [http://docs.opengeospatial.org/is/09-025r2/09-025r2.html#155].

### 6.3.5. Querying multiple feature collections in one query

WFS 3.0 Core only supports queries on a single feature collection (feature type in WFS 2.0, layers in

WMS/WMTS). In practice, it is often useful to query features from multiple collections at once, even if the query is restricted to simple filtering, e.g. `bbox` or `time`.

**Requirement "query-multiple-collections"**

The capability to select by default features from multiple collections in the dataset in a single request.

## 6.4. Organizing feature data, other representations

WFS 3.0 Core specifies the path `/collections/{collectionId}/items` to provide access to features on a by-feature basis.

For visualizing data in a map/scene viewer in a web browser, other approaches to organizing the feature data and other encodings are often better suited for the task.

**Requirement "query-additional-resource-types"**

The capability to make feature data, including complex data, available in bundles that are optimized for specific applications, for example, for streaming and visualization in a web browser.

The key example in the context of our use cases is the partitioning of space (2D or 3D). Typically, partitions are organized in some hierarchical order to support display of the features in each partition at different levels of detail (or "zoom level" in a 2D map view).

API extensions that support this requirement are likely not to be "WFS 3.0 extensions" as they are not about features as the main resources.

They would be API building blocks for other resource types, outside of the WFS 3.0 series, but based on a common API approach and a NextGen service architecture.

**Requirement "query-additional-encodings"**

The capability to make the data available in additional encodings.

Support for this is straightforward in the WFS 3.0 architecture (due to the use of HTTP and the architecture of the Web). Each encoding needs a media type, which should be registered with IANA [https://www.iana.org/assignments/media-types/media-types.xhtml].

# Chapter 7. Assessment of OGC standards and community specifications

## 7.1. General remarks

This section assesses current OGC and other specifications with respect to the identified requirements and analyses options for addressing the requirements in an architecture for a next generation of OGC services based on the approach taken by WFS 3.0.

This chapter is organized in three parts. First, we summarize how the different XML and JSON encodings match the requirements. Second, we look at different approaches to querying complex data and, third, we look at additional resource types and other ways to organize and encode the data.

## 7.2. Encodings

The following table provides an overview of how the different feature encodings for XML (GML, GML Simple Features Profile Level 0 and 2) and JSON (GeoJSON, CityJSON) match the requirements regarding data structures identified in the previous chapter.

CityJSON is, as the name implies, limited to 3D city models, provides a fixed schema and is an implementation of the model that underpins CityGML in JSON.

*Table 1. Comparison of XML and JSON encodings*

| **Requirement** | **GML, CityGML** | **GML-SF0** | **GML-SF2** | **GeoJSON** | **CityJSON** |
|---|---|---|---|---|---|
| multiple-values | Yes | No | Yes | Yes | Yes |
| nested-structures | Yes | No | Yes | Yes | Yes, where necessary |
| relationships | Yes | No | Yes | No | Yes |
| multiple-geometries | Yes | Yes | Yes | No | Yes |
| 3D-geometries | Yes | No | No | No | Yes |
| shared-geometries | Yes | No | No | No | No, only direct positions |
| non-linear-interpolation | Yes | Arcs only | Arcs only | No | No |
| schema-validation | Yes | Yes | Yes | Yes, but uncommon | Yes |

An issue with CityJSON is that there is no media type for CityJSON and the generic `application/json` media type would have to be used, which may create ambiguities with other potential JSON encodings for features in the future.

CityGML has no media type either, but CityGML is GML and the GML media type can be used. However, if there are multiple GML encodings for the same dataset, it would be necessary to register and use separate media types for each of the GML encodings. I.e., OGC should consider to register a media type for CityGML.

## 7.3. API building blocks for queries

Looking at extending the WFS 3.0 Core API with support for richer queries, there are three different approaches:

1. Use an existing approach used in other OGC standards with support for spatial queries,
2. use an existing approach used by the mainstream Web community that is well-known to developers and well-supported,
3. specify a new query mechanism that is simple to implement.

Filter Encoding and Common Query Language (CQL) belong to the first category, GraphQL and Falcor to the second category and the current discussions in the development of the SpatioTemporal Asset Catalog (STAC) [9] are in the third category.

This could potentially be extended with additional options, including (Geo)SPARQL, Gremlin, XQuery, etc. However, triple stores, graph databases or XML databases are not the typical database backends used when publishing spatial datasets for Web developers, so this analysis has focussed on the other options mentioned above.

Another option that was mentioned as a candidate in discussion is OpenSearch including the OpenSearch Geo/Time extensions [http://www.opengeospatial.org/standards/opensearchgeo]. After starting to analyze the OpenSearch Geo/Time extension this does not seem to be a real candidate. OpenSearch is more tailored towards full-text searches, overlaps with WFS 3.0 (paging, etc), is XML-based (OpenSearch descriptions, RSS/Atom as response formats) and the OpenSearch descriptions overlap with the OpenAPI definitions. OpenSearch is, therefore, not discussed in more detail.

To assess the options, the following aspects are considered:

- the capabilities to query spatial datasets based on the requirements identified in the previous chapter;
- the suitability to support different encodings, in particular XML and JSON;
- the support for popular database backends, mainly relational databases and document stores;
- support by libraries in key programming languages.

### 7.3.1. General considerations

This GitHub issue [https://github.com/opengeospatial/WFS_FES/issues/79] has additional discussion that should be considered.

During the discussions at the WFS 3.0 hackathon in March 2018 there was agreement that the path `/search/{endpoint}` should be reserved for offering richer queries on a dataset. Different search implementations could be published as separate search resources. i.e., a server could support

multiple types of queries in parallel. For example, `/search/fes` could be an endpoint for WFS 2.0 ad-hoc queries with Filter Encoding 2.0 and `/search/graphql` could be an endpoint for GraphQL queries on the same data.

In cases where only a query language is specified (e.g. WFS 2.0 ad-hoc queries with Filter Encoding 2.0, CQL, XQuery, etc.), but no rules exist how to execute queries via HTTP, there are three general options for the API design:

- Use of HTTP GET requests where the query is included in one or more query parameters.
- Use of HTTP POST requests where the query is included in one or more query parameters in the payload with a media type of `application/x-www-form-urlencoded`.
- Use of HTTP POST requests with the query in the payload to create a new query resource on the server. The request would return the URI of the new query. Accessing the query URI with HTTP GET would return the result of the query. HTTP PUT and DELETE could be used to update or delete the query.

The third option is quite heavy for queries that are not used repeatedly. For queries that are executed more than once, however, this approach has benefits, e.g. for caching. The approach would also support parameterized queries where the parameters could be passed as query parameters when accessing the query resource with a HTTP GET request (like stored queries in WFS 2.0).

### 7.3.2. WFS 2.0 ad-hoc queries and Filter Encoding 2.0

The OGC Filter Encoding Standard 2.0 [10] is the standard filter language for ad-hoc queries in WFS 2.0 [11]. Together they support the query requirements identified in the previous chapter.

However, the specifications are closely tied to XML including the use of XPath, which makes them an unnatural match for servers that want to return JSON.

Implementations exist, but the requirements identified in the previous chapter are supported only to a limited extent in freely available libraries. Implementations typically support relational database backends.

**Conclusion: A candidate query language for WFS 3.0 implementations that also support WFS 2.0 and GML. Complex feature handling in WFS 3.0 will, of course, only be supported, if the WFS 2.0 implementation already supports the identified use cases.**

### 7.3.3. CQL

CQL is short for "OGC Common Query Language" and is defined in the OGC Catalogue Services Standard 2.0 [12] in section 6.2.2. It is a text-based syntax similar to the SQL "Where" clause, i.e. on a similar level as Filter Encoding, but easier to write for most developers than the XML syntax of Filter Encoding.

| NOTE | The Catalog Service 3.0 also includes a syntax definition for CQL, but lacks any rules or requirements for it. This report therefore references version 2.0 of the standard. |
|---|---|

Implementations typically support relational database backends, CQL is less suited for document

stores.

As a text-based query language it is not strongly tied to XML, JSON or any other encoding.

CQL does not support all identified query requirements. CQL does not have well-defined semantics for querying properties with multiple values, does not support link traversal across relationships (query predicates on related features) or 3D geometries.

Implementations exist, but the requirements identified in the previous chapter are only supported to a limited extent in freely available libraries. An example is limited support for "compound attributes", i.e. path expressions that might be used to query nested data structures.

Beside the fact that implementations support only a subset of the language, CQL has a major limitation with respect to the requirements identified in the previous chapter: Attribute values are assumed to be literal. That is, CQL does not support values that are collections (maximum multiplicity > 1) or objects / data types.

That is, either the use of CQL is restricted to feature data that meets these constraints or CQL would need to be extended to support feature data that have the following data structure requirements:

- "multiple-values"
- "nested-structures"
- "relationships"

| | |
|---|---|
| **NOTE** | Queries using solid geometries or non-linear curve interpolations are not supported in CQL, but at least the use cases did not identify this as an important requirement. |

For nested structures and relationships, probably the dot-notation the compound attribute names could be used, but for each encoding the mapping between the compound attribute names and the feature encoding would need to be specified.

For predicates on attributes with multiple values, there are basically two options:

- Add comparison operators that support collections and not just literal values.
- Follow the same approach as Filter Encoding 2.0 with the matchAction [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html#50] parameter (where the default is that an expression evaluates to `true`, if at least the expression evaluates to `true` for at least one of the attribute values).

In addition, the specification for time period expressions in CQL should be amended to cover the cases discussed in the related WFS 3.0 issue [https://github.com/opengeospatial/WFS_FES/issues/155].

In order to support clients to construct queries, the feature properties that may be queried should be enumerated for each feature type. This could be included in the feature collection metadata or, which is probably preferable, it could be made available in an additional resource listing all queryable properties. For example at `/collections/{collectionId}/queryables`. The result could be a JSON object with a member for each property of the feature. The value of the member could be used to identify the data type. If the property value is a related object in the dataset, the queryables resource of that collection could be referenced. For nested objects, the compound attribute values

could be used explicitly.

It should also be allowed to declare queryables that do not have to be, for example, a direct member of the `properties` object in a GeoJSON feature.

CQL is currently "buried" in the Catalogue Service specification and the specification of the language is largely restricted to the definition of the grammar (with a number of inconsistencies and ambiguities in the definition). If CQL would be supported by a WFS extension, it should be extracted from the Catalogue Service standard and become a standard on its own, with a clear and unambiguous specification of the language and requirements for implementations.

**Conclusion: A candidate query language for an intermediate WFS 3.0 query capability that goes beyond the limited support that WFS 3.0 Core offers, but that may not address all requirements identified. The lack of implementation support for document stores is an issue that needs a broader discussion, too.**

#### Supporting other requirements

There are three other requirements that are out-of-scope for CQL and which would need to be covered by additional WFS 3.0 extensions:

- Requirement "embed-related-features": To add related features to the response, another mechanism would have to be specified, independent of the CQL query. For example, a query parameter `resolve` could be specified with a list of feature-valued properties where the values should be included in the response, if the related features are in the same dataset.
  This capability would be similar to the `resolvePath` attribute in WFS 2.0 with `resolve=local`.

- Requirement "return-subset": To return only some of the properties another query parameter, e.g. `properties`, could be used with a list of properties that should be returned. The mapping between the names of the properties in the query parameter and the feature encoding needs to be specified for each encoding. In GeoJSON, for example, `id` and `geometry` are not part of the properties JSON object, but should be supported, too. Similar in CityJSON where the address information, the geometry or links to child features are encoded separately from the attributes JSON object.
  This capability would be similar to the `propertyName` query parameter in the WFS 2.0 KVP encoding.

- Requirement "query-multiple-collections": In addition to the `/collections/{collectionId}/items` endpoint for each sub-collection in a dataset, an additional `/items` endpoint could be added that provides access to features from all the sub-collections. A query parameter `collections` could be added to access only features from the listed collections in the response.

In the `resolve` and `properties` parameters, the same notation should be used for properties of data types or related features as it is used by the compound attribute names in CQL queries (see the discussion above).

#### Examples

Let’s assume

- that CQL would be extended as described above,
- that the query parameters `resolve` and `properties` would be supported as described above,
- that a CQL query would be expressed on an `items` resource in a query parameter `where` and
- that an additional query parameter `where-lang` is used to support multiple languages for `where` predicates (the examples below use `cql` for simpler expressions without compound attributes etc. and `cql-extended` for expressions that require extensions beyond a CQL Core as described above).

| NOTE | Since CQL does not define a full query, but a "where" clause on a feature collection, a separate `/search/cql` (or similar) endpoint is not necessary. |
|---|---|

The queries in the cadastral use cases could be expressed as follows, assuming a GeoJSON feature encoding.

Selection of protected sites:

*Query 1*

```
/collection/ProtectedSite_Water/items?
where-lang=cql-extended&
where=contains.geometry INTERSECTS ENVELOPE(7.0244,7.1296,50.5351,51.4362)
```

*Query 2*

```
/collection/ProtectedSite_Water/items?
where-lang=cql-extended&
where=contains.geometry INTERSECTS ENVELOPE(7.0244,7.1296,50.5351,51.4362)&
resolve=contains
```

Select the owners of cadastral parcels in an area:

*Query*

```
/collection/Persons/items?
where-lang=cql-extended&
where=partOf.relatedTo.geometry INTERSECTS ENVELOPE(7.0348,7.0452,50.6252,50.7154) OR
      partOf.related.relatedTo.geometry INTERSECTS
ENVELOPE(7.0348,7.0452,50.6252,50.7154)
```

The queryables at `/collection/{collectionId}/queryables` could be the following JSON objects:

*Queryable properties of the features in the collection "Persons"*

```
{
  "firstName": "String",
  "lastName": "String",
  "partOf": "http://example.com/cadastre/collection/Records/queryables",
  "lifespan.begin": "DateTime",
  "lifespan.end": "DateTime"
}
```

*Queryable properties of the features in the collection "Records"*

```
{
  "recordId": "String",
  "right": "String",
  "related": "http://example.com/cadastre/collection/Records/queryables",
  "relatedTo": "http://example.com/cadastre/collection/CadastralParcels/queryables",
  "lifespan.begin": "DateTime",
  "lifespan.end": "DateTime"
}
```

*Queryable properties of the features in the collection "CadastralParcels"*

```
{
  "parcelId": "String",
  "geometry": "Geometry",
  "lifespan.begin": "DateTime",
  "lifespan.end": "DateTime"
}
```

Select versions of cadastral parcels based on their temporal validity:

*Query*

```
/collection/CadastralParcels/items?
where-lang=cql-extended&
where=lifespan.begin DURING 2017-07-01T00:00:00Z/2017-07-01T23:59:59Z
```

**NOTE** The request could simply use `time=2017-07-01T00:00:00Z/2017-07-01T23:59:59Z`, too, which is already possible with WFS 3.0 Core.

Select cadastral parcels for rendering with a specific style:

*Query*

```
/collection/CadastralParcels/items?
where-lang=cql-extended&
where=denominator IS NOT NULL AND
      (altLegalStatus IS NULL OR altLegalStatus=false) AND
      textOnMap.type = 'ZAE_NEN'
```

**NOTE** This ssumes that the interpretation of `textOnMap.type = 'ZAE_NEN'` is that at least one such value exists.

Selection of building parts of a building:

*Query*

```
/collection/CadastralParcels/items/DENW45AL0000lxrJ?
resolve=contains
```

**NOTE** This requires that the topological relationship would be included in the feature data explicitly.

The feature queries in the 3D / heating demand use cases could be expressed as follows, assuming a CityJSON feature encoding.

Using WFS to query the simulation result and visualize it in a 3D scene by building Id:

*Query*

```
/ny_dataset/collection/buildings/items/uuid_2824afd6-00e5-42ac-ab95-ec868595dc5a?
properties=function,measuredHeight,heat
```

Query a feature from a city model by id:

*Query*

```
/my-city-model/collections/buildings/items/TWINHOUSE1
```

Select buildings in a 2D region from a city model:

*Query*

```
/my-city-model/collections/buildings/items?bbox=-74,40.7,-73.96,40.8
```

Select buildings based on nested features or properties:

*Query 1*

```
/my-city-model/collections/buildings/items?
where-lang=cql-extended&
where=geometry.lod2 INTERSECTS POLYGON(...)
```

**NOTE** This assumes that the attribute `geometry.lod2` would map for a CityJSON encoding to the feature geometry with `lod: 2`

*Query 2*

```
/my-city-model/collections/buildings/items?
where-lang=cql-extended&
where=address.ThoroughfareName LIKE 'Unter den Linden%'
```

**NOTE** This assumes that the attribute `address.ThoroughfareName` would map for a CityJSON encoding to the `ThoroughfareName` member in the `address` JSON object).

The other queries from the use case are similar and are skipped here. In general, a pre-requisite is how attribute names map to the content of the CityJSON encoding.

**NOTE** The 3DPS requests from the use cases are addressed here.

### 7.3.4. Falcor

Falcor [13] is a data platform that powers the Netflix user interfaces.

The starting point of Falcor is to assume that all data is a single (virtual) JSON object. This allows clients to work with the data using standard operations on JSON objects and support for Path expressions. In the words of Netflix: "If you know your data, you know your API" [14].

In a way this would be comparable to using XQuery as a query language in WFS 2.0 where the dataset is basically a large GML feature collection.

Falcor has additional conventions to allow that the virtual JSON object can be used as a graph with shared resources and not just the that a JSON object is. This avoids multiple copies of the same object in different parts of the virtual JSON object. An enhanced path notation is used to reference nodes within the virtual JSON object.

Falcor has no schema of the data and assumes that the developer knows the data (see the quote above).

It is mainly designed for use in JavaScript and has no support for geometries or spatial predicates.

**Conclusion: Falcor may be a candidate for a WFS 3.0 implementation that only supports JSON and that is mainly accessed from JavaScript. However, support for spatial aspects would need to be specified and implemented first. It is thus not considered in more detail in this report.**

### 7.3.5. GraphQL

GraphQL [15] is a declarative, string-based query language created by Facebook to support fetching data for use in a user application from a server.

| NOTE | A similarity with Filter Encoding is the intent to be a declarative language independent of the underlying database technology. |
|---|---|

One of the main drivers for GraphQL was the goal to provide an interface that allows mobile app developers to retrieve exactly the data that they need in a single query from a single endpoint. This is based on the observation that in REST APIs one usually needs multiple requests to fetch the information and/or that the response often contains unnecessary information ("overfetching").

That is, support for GraphQL would basically be complementary to the current WFS 3.0 Web API. The blogpost "GraphQL: Everything You Need to Know" [16] includes a comparison of strengths and weaknesses of both approaches.

| NOTE | A GraphQL endpoint could be implemented on top of the Web API, but likely with sub-optimal performance. |
|---|---|

Unlike Falcor, where the client has to know the data, GraphQL **requires** a schema of the data. GraphQL is strongly typed and supports nesting, multiplicities, etc.

Typically, GraphQL schemas are tailored for the specific application needs. That is, GraphQL queries are in practice in a way closer to the stored queries of WFS 2.0 than the generic ad-hoc queries of WFS / FES 2.0 - although with a much richer mechanism to specify parameters and projection clauses.

A significant plus for GraphQL is that it has a lively, and growing, ecosystem with good tools, support, etc.

However, currently there is no support for geometries or spatial queries in GraphQL.

GraphQL is not tied to JSON, but JSON seems to be by far the most commonly used encoding.

**Conclusion: GraphQL is a promizing candidate because of its popularity and its characteristics, in particular for usages that are close to end user applications. Spatial support may be an issue and needs to be explored in more detail, including the use of GeoJSON or CityJSON.**

#### An example query

We will use the data from one of our use cases to explore how feature queries could be supported using GraphQL.

The first step is to define the schema of the relevant data. In general, we could take two different approaches:

- define the schema so that the data returned by the queries contains valid GeoJSON consistent with the feature data provided by the WFS; or

- define the schema so that it is tailored towards the query needs.

The first approach is closer to WFS and simplifies the reuse of the query results in contexts where GeoJSON objects can be processed directly. The second approach is closer to pre-defined stored queries, with new data structures derived from the feature data.

In this testbed, only the first approach is explored (since in the second approach it is difficult to see what could be specified in a WFS-related standard).

Here is how a GraphQL schema could look like for the features in the use case:

*GraphQL schema for Person, Record and CadastralParcel features*

```
interface GeoJSONObject {
  id: ID!
  type: GeoJSONType!
}

enum GeoJSONType {
  Point
  MultiPoint
  LineString
  MultiLineString
  Polygon
  MultiPolygon
  GeometryCollection
  Feature
  FeatureCollection
}

type Person implements GeoJSONObject {
  id: ID!
  type: GeoJSONType!
  properties: PersonProperties
}

type PersonProperties {
  firstName: String
  lastName: String
  partOf: [Record]
}

type Record implements GeoJSONObject {
  id: ID!
  type: GeoJSONType!
  properties: RecordProperties
}

type RecordProperties {
  right: RightType
  relatedTo: [CadastralParcel]!
```

```
}

enum RightType {
  SomeRight
  AnotherRight
  YetAnotherRight
}

type CadastralParcel implements GeoJSONObject {
  id: ID!
  type: GeoJSONType!
  geometry: GeoJSONMultiSurface
  properties: CadastralParcelProperties
}

type GeoJSONMultiSurface {
  type: GeoJSONType!
  coordinates: [[[[Float]]]]
}

type CadastralParcelProperties {
  parcelNumber: String
  area(unit: AreaUnit = m2): Float
  records: [Record]
}

enum AreaUnit {
  m2
  ft2
}

# the schema allows the following queries:
type Query {
  parcels: [CadastralParcel]
  person(id: String!): Person
}
```

Let’s assume we have the following data instances:

*Sample data*


```
{
  "persons": [
    { "type": "Feature", "id": "1", "properties": { "firstName": "John", "lastName":
"Doe", "partOf": [ "1" ] } },
    { "type": "Feature", "id": "2", "properties": { "firstName": "Erika", "lastName":
"Mustermann", "partOf": [ "2", "3", "4" ] } },
  ],
  "records": [
    { "type": "Feature", "id": "1", "properties": { "right": "SomeRight", "relatedTo":
[ "DENW19AL0000geMFFL" ] } },
    { "type": "Feature", "id": "2", "properties": { "right": "SomeRight", "relatedTo":
[ "DENW19AL0000genyFL" ] } },
    { "type": "Feature", "id": "3", "properties": { "right": "AnotherRight",
"relatedTo": [ "DENW19AL0000geqyFL" ] } },
    { "type": "Feature", "id": "4", "properties": { "right": "YetAnotherRight",
"relatedTo": [ "DENW19AL0000geqyFL", "DENW19AL0000ger1FL" ] } }
  ],
  "parcels": [
    { "type": "Feature", "id": "DENW19AL0000geMFFL", "properties": { "parcelNumber":
"193", "area": 1739.0 }, "geometry": { "type" : "MultiPolygon", "coordinates" : [ [ [
[ 8.711910494386446, 51.49108376876667 ], [ 8.71229996279325, 51.491067136843 ], [
8.71238480759279, 51.49164525475229 ], [ 8.712013831642468, 51.491666041604 ], [
8.711993473544684, 51.4916028694182 ], [ 8.711960339932295, 51.49139558179017 ], [
8.711953300233393, 51.49135154980474 ], [ 8.711910494386446, 51.49108376876667 ] ] ]
]} },
    { "type": "Feature", "id": "DENW19AL0000genyFL", "properties": { "parcelNumber":
"174", "area": 4533.0 }, "geometry": { "type" : "MultiPolygon", "coordinates" : [ [ [
[ 8.697513007025446, 51.50144133282769 ], [ 8.697820179888327, 51.50128890506469 ], [
8.699285444473853, 51.50252385737269 ], [ 8.698973376983412, 51.50259426261686 ], [
8.698937901399875, 51.5026408567456 ], [ 8.698295428101916, 51.50210001561205 ], [
8.697513007025446, 51.50144133282769 ] ] ] ] } },
    { "type": "Feature", "id": "DENW19AL0000geqyFL", "properties": { "parcelNumber":
"74", "area": 10175.0 }, "geometry": { "type" : "MultiPolygon", "coordinates" : [ [ [
[ 8.686673298571343, 51.50079992000421 ], [ 8.68679160662328, 51.50074125004235 ], [
8.687436267676059, 51.50042151431735 ], [ 8.688147581070762, 51.500903657307006 ], [
8.688887649073012, 51.50140520447189 ], [ 8.688883177902552, 51.50143555777073 ], [
8.688149161509205, 51.50179930330681 ], [ 8.68738593229976, 51.50128249372289 ], [
8.686673298571343, 51.50079992000421 ] ] ] ] } },
    { "type": "Feature", "id": "DENW19AL0000ger1FL", "properties": { "parcelNumber":
"103", "area": 6894.0 }, "geometry": { "type" : "MultiPolygon", "coordinates" : [ [ [
[ 8.688736032883249, 51.5036400173296 ], [ 8.689168817396284, 51.50342556070846 ], [
8.689822053323931, 51.50386836226231 ], [ 8.690374046734465, 51.504242519669305 ], [
8.690927010128721, 51.50461732440779 ], [ 8.690749468566104, 51.50470536227641 ], [
8.690494250855838, 51.50483185947567 ], [ 8.689649601209783, 51.50425930806056 ], [
8.688736032883249, 51.5036400173296 ] ] ] ] } }
  ]
}
```

The following GraphQL query would return selected information about the parcels on which the person with id "2" has rights.

*Sample GraphQL query*

```
query ParcelsForPerson {
  person(id: "2") {
    properties {
      firstName
      lastName
      partOf {
        properties {
          right
          relatedTo {
            id
            properties {
              area
              parcelNumber
            }
          }
        }
      }
    }
  }
}
```

The query requests the following information in the result:

- the first and last name of the person;
- the right for each cadastral record that is associated with the person;
- the parcels for each of the cadastral records (the identifier, the parcel number and the parcel area).

All other properties, including the geometries are suppressed in the query result.

*Query result*

```
{
  "data": {
    "person": {
      "properties": {
        "firstName": "Erika",
        "lastName": "Mustermann",
        "partOf": [
          {
            "properties": {
              "right": "SomeRight",
              "relatedTo": [
                {
                  "id": "DENW19AL0000genyFL",
```

```
                "properties": {
                  "area": 4533,
                  "parcelNumber": "174"
                }
              }
            ]
          }
        },
        {
          "properties": {
            "right": "AnotherRight",
            "relatedTo": [
              {
                "id": "DENW19AL0000geqyFL",
                "properties": {
                  "area": 10175,
                  "parcelNumber": "74"
                }
              }
            ]
          }
        },
        {
          "properties": {
            "right": "YetAnotherRight",
            "relatedTo": [
              {
                "id": "DENW19AL0000geqyFL",
                "properties": {
                  "area": 10175,
                  "parcelNumber": "74"
                }
              },
              {
                "id": "DENW19AL0000ger1FL",
                "properties": {
                  "area": 6894,
                  "parcelNumber": "103"
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```


The example has been set up in an Apollo GraphQL Launchpad and you can play with different

queries: https://launchpad.graphql.com/07v1j3zzm5.

From the example, we can derive some topics for discussion and further experimentation:

- The use of the `properties` object in GeoJSON results in quite complex schemas and results. It would be more natural in the queries, if the properties would be members of the feature object instead of being nested in a sub-object.
- To be a valid GeoJSON feature, the JSON object must have a `geometry` member, which may be `null`. It is not possible to simply drop the member from the result. As a consequence, the feature objects in the result above are not valid GeoJSON features.
- GeoJSON-aware software probably does not expect nested GeoJSON features anyhow.

It is, therefore, questionable, if there is value in using GeoJSON in the GraphQL schema. It seems quite likely that a tailored GraphQL schema for the specific query needs would be better suited. This requires more experiments and further work.

With CityJSON as an encoding this should be even more of an issue due to the distributed way in which geometry and properties are encoded across the JSON document.

### 7.3.6. Queries in the SpatioTemporal Asset Catalog (STAC)

The SpatioTemporal Asset Catalog (STAC) specification [9] intends to standardize the way geospatial assets are exposed online and queried. The specification defines a **spatiotemporal asset** as "any file that represents information about the earth captured in a certain space and time". Right now, the focus is on remotely-sensed imagery.

Querying STAC is very similar to general feature querying and since the principles and technologies used are very similar, the WFS 3.0 hackathon in March 2018 [17] was co-located with a STAC sprint [18]. During these meetings the API building blocks were aligned so that STAC implementations will conform to WFS 3.0 Core.

STAC extends the Core with a `/search/stac` endpoint, which for now is restricted to bounding box and time interval searches like WFS 3.0 Core.

Several ideas are discussed or explored for supporting more advanced queries:

- The STAC roadmap [https://github.com/radiantearth/stac-spec/blob/master/roadmap.md#querying-and-filtering] mentions ECQL [https://github.com/geotools/geotools/blob/master/modules/library/cql/ECQL.md] - a variant of CQL ("a CQL Like language") implemented in GeoTools - and Backand NOSQL query language [http://backand-docs.readthedocs.io/en/latest/apidocs/nosql_query_language/index.html] as options.
- In Fort Collins, a JSON encoding of a query filter was presented and discussed. This development continues, see these OpenAPI fragments for a query extension [https://github.com/radiantearth/stac-spec/blob/dev/api-spec/extensions/query.fragment.yaml].

| | |
|---|---|
| **NOTE** | Another STAC extension related to the use cases is a capability to shape the feature properties to be included in the response [https://github.com/radiantearth/stac-spec/blob/dev/api-spec/extensions/fields.fragment.yaml]. |

**Conclusion: Before any decision is made for WFS 3.0 query extensions, the plans should be discussed with the STAC community to check for additional opportunities to align the specifications.**

### 7.3.7. Summary

The following table summarizes in how far the candidates support the identified requirements.

*Table 2. Comparison of candidate query languages*

| Requirement | WFS 2.0 + FES | CQL | GraphQL | STAC JSON |
|---|---|---|---|---|
| query-properties-with-multiple-values | Yes | No | Yes | No |
| query-nested-properties | Yes, but limited support in implementations | Yes, but limited support in implementations | Yes | No |
| query-logical-operators | Yes | Yes | Yes | Yes |
| query-2D-geometries | Yes | Yes | No | Yes |
| query-buffers | Yes | Yes | No | No |
| query-3D-geometries | Yes | No | No | No |
| query-current-data | n/a | n/a | Yes, this can be implemented in the query definitions | n/a |
| embed-related-features | Yes | n/a, but see here | Yes | No |
| return-subset | Yes, but with limitations | n/a, but see here | Yes | Yes |
| query-multiple-collections | Yes | n/a, but see here | Yes | Yes |

## 7.4. API building blocks for additional resource types

### 7.4.1. Tiles (2D)

For 2D data, a commonly used approach is to organize the feature data in tiles, in particular for visualization in map-based client applications in a web browser. Tiles are provided for different zoom levels (scales) and how the features that are located in the bounding box of a tile are included in the tile will depend on the zoom level (e.g. no buildings at a scale of 1:1.000.000).

In parallel to OGC Testbed 14 another OGC Innovation Program initiative, the Vector Tiles Pilot, is investigating how tiled vector data should be provided via a NextGen service Web API as Mapbox Vector Tiles (using Google Protocol Buffers) and as GeoJSON. Of course, the tiles could also be rendered as bitmap images, too, if the server has styling information.

In general, as tiles are different resources, they would be made available under new resource paths. For example:

- `/collections/{collectionId}/tiles/{tilingScheme}/{zoomLevel}/{row}/{column}`: Tiles with features of a single collection.
- `/tiles/{tilingScheme}/{zoomLevel}/{row}/{column}`: Tiles with feature data from multiple collections / with multiple layers. Like in the case of the `/items` path proposed above, a query parameter `collections` should be added to access only features from a selected list of collections in the response.

These paths (in addition to paths for the tiling scheme information) represent the capabilities of an OGC WMTS in a NextGen architecture.

In addition, Google Protocol Buffers following the Mapbox Vector Tile format could also be served from the `/collections/{collectionId}/items` path as an additional encoding. A pre-requisite is a media type for the encoding to support content negotiation.

### 7.4.2. Scenes (3D)

A common approach to provide optimized access to 3D feature data for visualization in a browser are "scenes". A scene provides 3D geometries with texture data and attribute information, organized as a scene graph and/or spatial index. Each node in the graph represents a spatial partition and data for display at a certain level of detail, depending on the distance from the viewpoint, etc.

The OGC 3DPS standard [19] provides access to scenes, usually using the OGC community specifications i3s [20] and 3D Tiles [21].

A difference to the 2D tiles case described in the previous section is that it is the client that requests tiles for display based on the knowledge of the tiling scheme. In the 3DPS case, the nodes in the scene graph are secondary resources, linked from the scene graph. That is, the client accesses a scene graph and then access the nodes linked from the graph.

A possible implementation for fetching scenes of a dataset in a NextGen architecture could be

- `/collections/{collectionId}/scene`: A scene with features of a single collection.
- `/scene`: A scene with feature data from multiple collections / with multiple layers. Again, a query parameter `collections` should be added to access only features from a selected list of collections in the response (in 3DPS 1.0: `layers`).

| | |
|---|---|
| **NOTE** | It is important to understand the difference to the `tiles` paths above. These represent enumerable resources (the tiles). The `scene` paths are different, they represent a single processing resource that derives a scene from the collection or dataset based on the requested characteristics. Therefore, plural is used for `tiles` and singluar for `scene`. |

| | |
|---|---|
| **NOTE** | Supporting WMS capabilities in the NextGen architecture could follow the same approach, i.e. to support `/collections/{collectionId}/map` and `/map` resources. |

The `scene` paths should support the usual WFS 3.0 query parameters:

- `bbox` (in 3DPS 1.0: `boundingbox`),
- `time` (in 3DPS 1.0: not supported),
- `crs` (from the WFS 3.0 CRS extension, in 3DPS 1.0: `crs`)
- `bbox-crs` (from the WFS 3.0 CRS extension, in 3DPS 1.0: part of `boundingbox`)
- `properties` (proposed above, for scene encodings that support feature properties, in 3DPS 1.0: not supported)
- `where` (proposed above, in 3DPS 1.0: not supported)

The last two parameters would be up for discussion, but in general it should be helpful for clients, if feature selection is done consistently across the different resources in the API.

In addition, the other parameters of the GetScene request (beside `request`, `version` and `format` which are no longer needed as this is handled differently) would be supported, too. For example, `lods` or `styles`.

Steinbeis Transfer Center at HFT Stuttgart (Steinbeis) has implemented an experimental 3D Portrayal Service scene resource following the approach based on the Testbed 13 showcase using the 3D CityGML model of New York described here [http://docs.opengeospatial.org/per/17-046.html]. The implementations is based on OpenAPI and the Web API can be tested at http://steinbeis-3dps.eu:8080/tb14/API/index.html [http://steinbeis-3dps.eu:8080/tb14/API/index.html].

Alternatively, to use the Web API call in an application, make the following request by defining the correct `bbox` coordinates in the request:

```
 http://steinbeis-
3dps.eu:8080/tb14/wfs3/3D_CityModel_manhattan/collections/buildings/scene?format=appli
cation/json&bbox=-73.99562241015136,40.72227595290325,-
73.9799164993194,40.749273379993284
```

The aforementioned call will deliver a 3D building model (as one layer of the 3D city model (3D-DLM)) of Manhattan using 3D Tiles. It can be rendered in the Cesium globe (See Figure 11.). A prototype implemetation is available at http://81.169.187.7:8081/3DPS_App/.

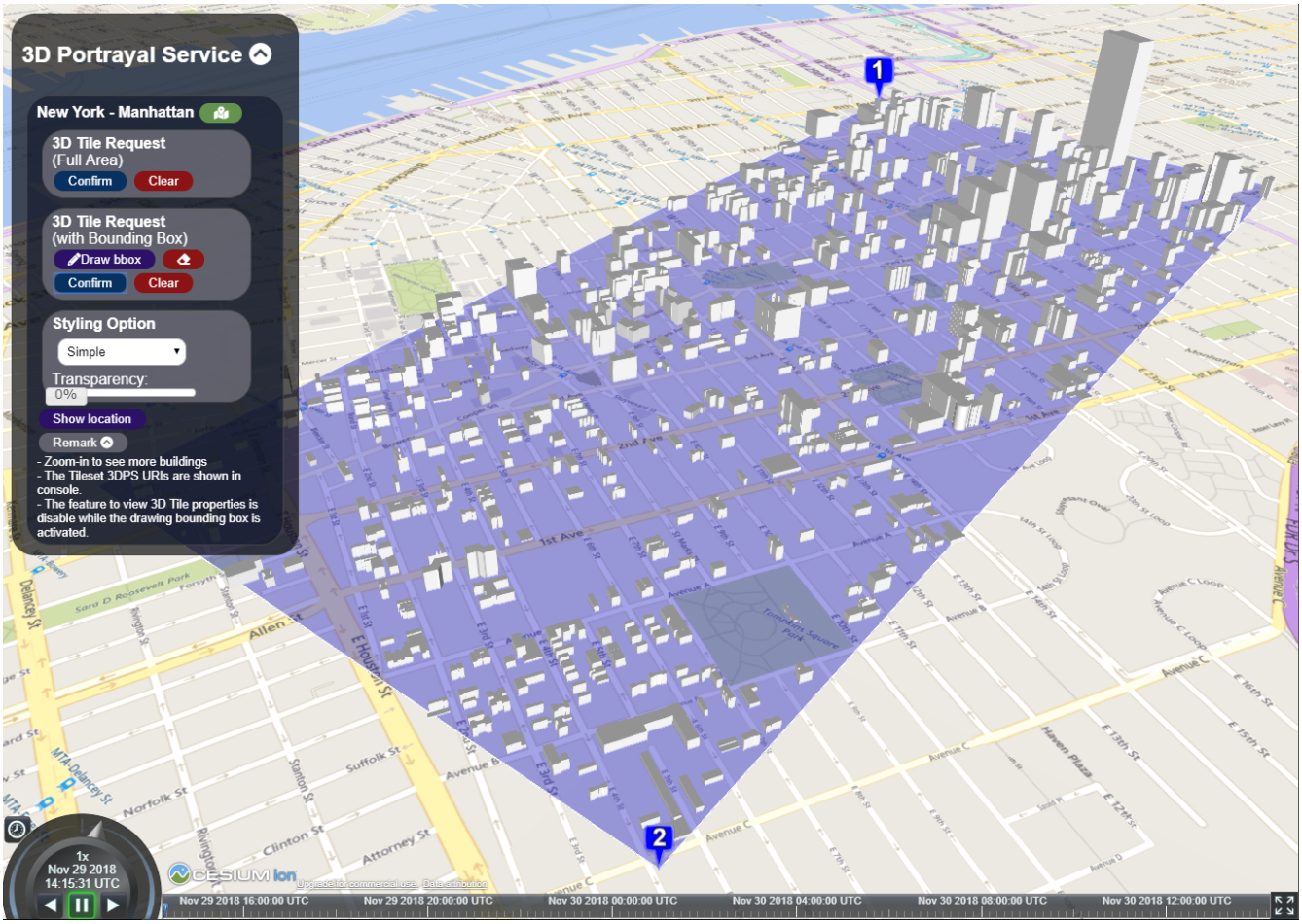


*Figure 11. Cesium Globe 3D Tiles Manhattan*

The current OpenAPI implementation supports two resources **Data Layer** and **3D Scene** (See Figure 12.).

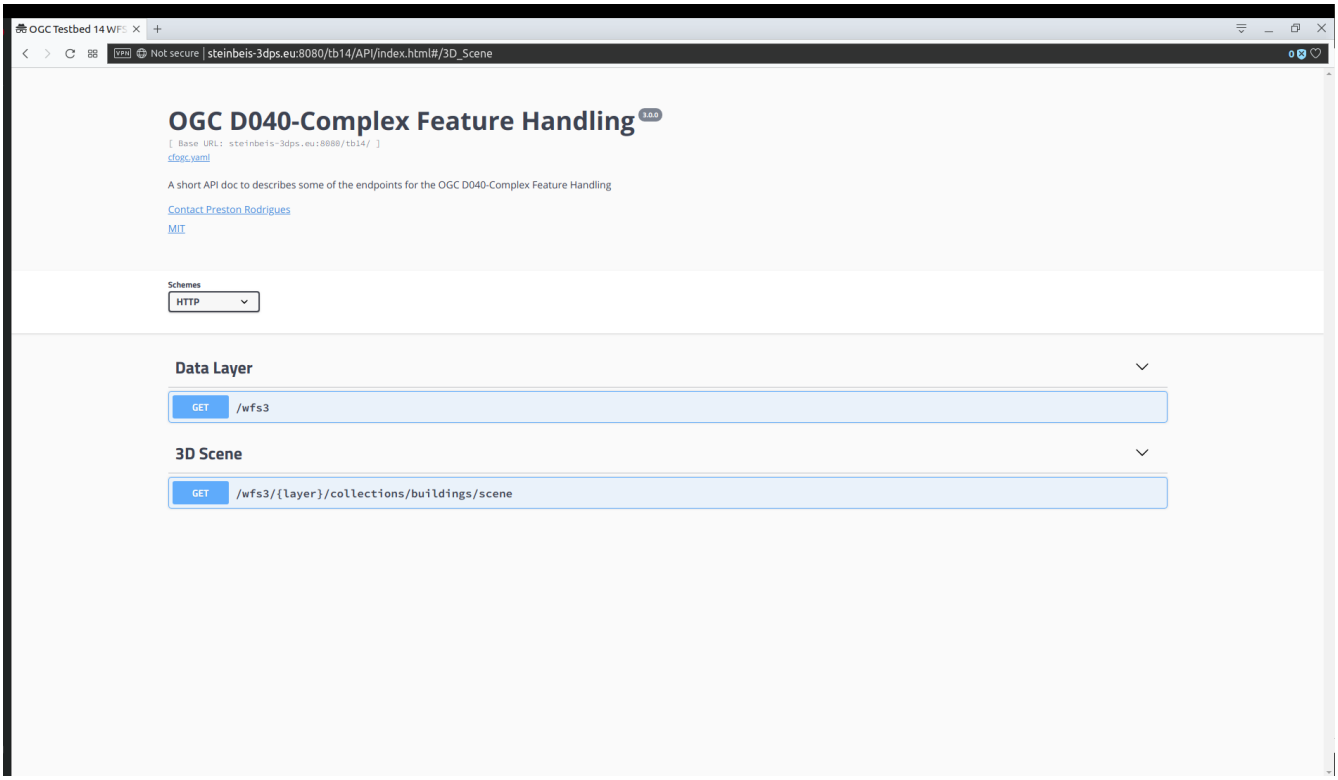


*Figure 12. OpenAPI definition*

The **data Layer** resource is used to retrieve the name of the available data layers. The current implementation supports one layer only: *“3D_CityModel_manhattan”* . However, in case the service supports more then one data layer it will show a list of supported data layers.

| NOTE | **data Layer** will be changed to **dataset** in the next update. |
|---|---|

```
{ "datalayer": "3D_CityModel_manhattan,3D_CityModel_geneva,3D_CityModel_stuttgart" }
```

| NOTE | A simplified approach is taken in the prototype. |
|---|---|

To test the Web API, first click on the *Try it out* button and then the *Execute* button.

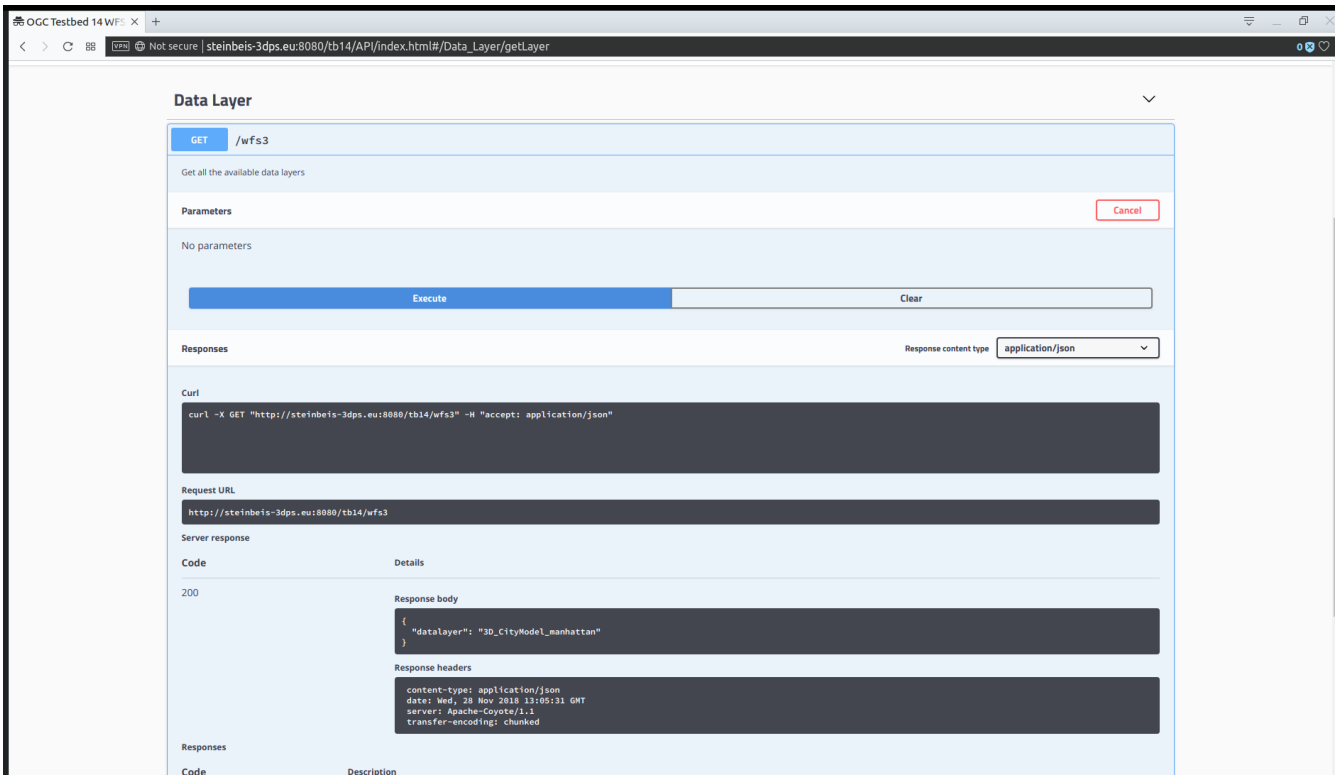


*Figure 13. OpenAPI Data Layer*

Since this call does not take any input you will see the output as shown in Figure 13. In addition, it also shows the curl request with all the necessary query parameters.

The **3D Scene** function on the other hand takes three input parameters. The first is a path parameter (*layer*) and the remaining two are query parameters (*format*,*bbox*) see Figure 14.

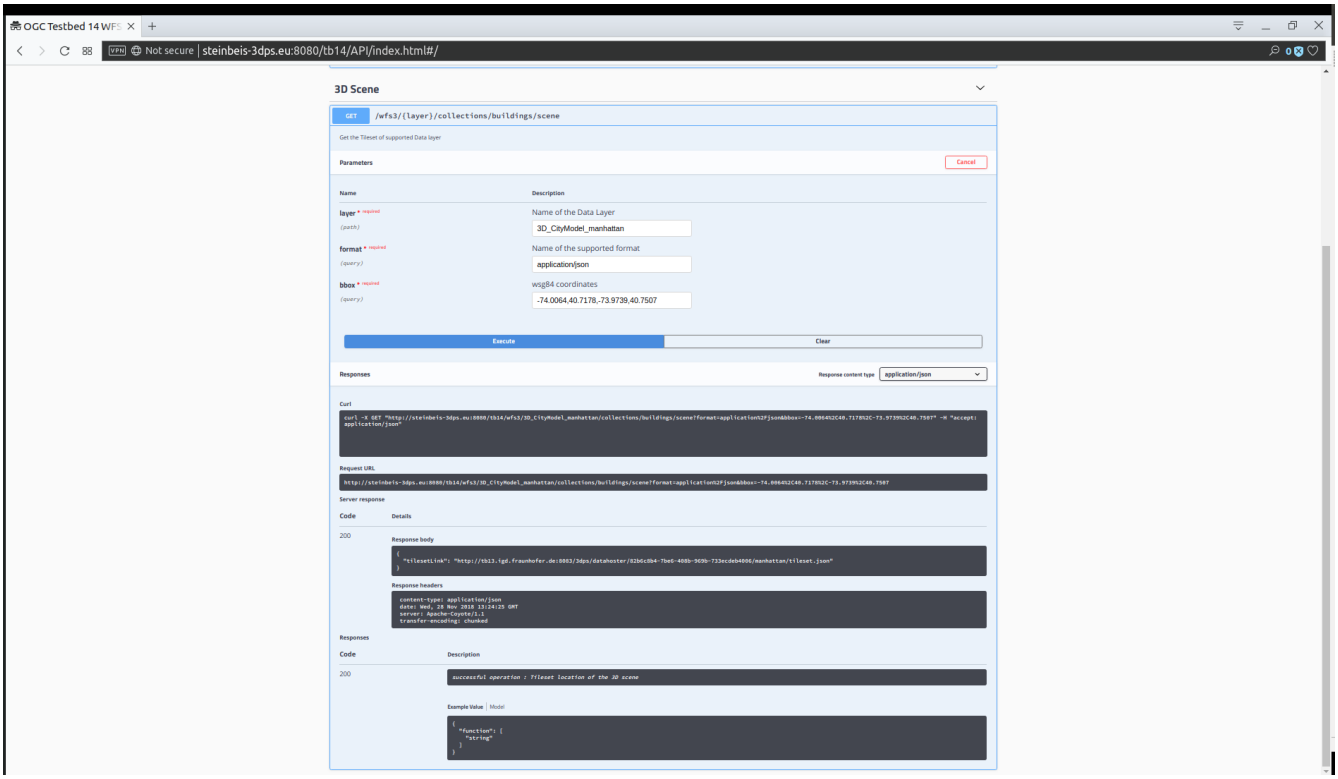


*Figure 14. OpenAPI 3D Scene*

A successful execution will provide the tileset link (see example below).

```
"tilesetLink": "http://tb13.igd.fraunhofer.de:8083/3dps/datahoster/fc38b549-4989-4703-bb27-f5b4fd33fd90/manhattan/tileset.json"
```

**NOTE** A simplified approach is taken in the prototype.

This *tilesetLink* can be rendered in the Cesium globe. To show the feasibility of our approach we have developed an integrated prototype see Figure 15. Please visit the following http://81.169.187.7:8081/3DPS_App/ [http://81.169.187.7:8081/3DPS_App/].

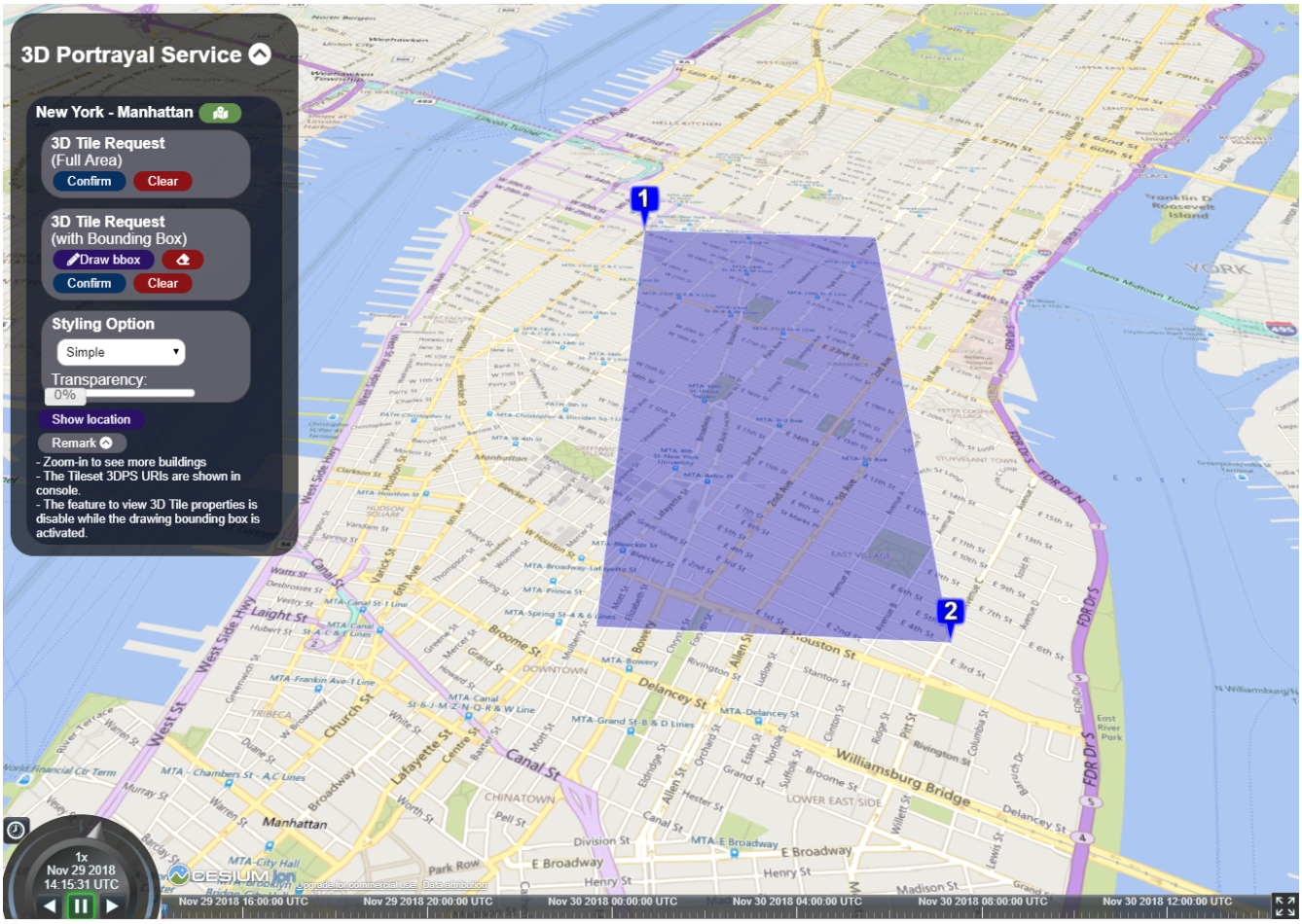


*Figure 15. 3DPS APP*

To learn more and get used to the platform please visit the you tube link https://www.youtube.com/watch?v=6EwlwAhtK7Q [https://www.youtube.com/watch?v=6EwlwAhtK7Q].

To include multiple layers (for example, buildings and vegetation), the request would be:

```
../3D_CityModel_manhattan/scene?collections=buildings,vegetation&bbox=-
74.0064,40.7178,-73.9739,40.7507
```

**NOTE** Multiple layers are not implemented in the prototype.

The information that is currently included in 3DPS 1.0 capabilities would be included in the OpenAPI document and in extensions to the feature collection metadata resources like in WFS 3.0.

The selection of an encoding would follow the same approach as in WFS 3.0 [https://rawgit.com/opengeospatial/WFS_FES/master/docs/17-069.html#_encodings_2]. That is, every server will support content negotiation using media types to negotiate the format of the response. In addition, servers should support a mechanism to include the format information in the path to support hyperlinks.

Media types for i3s and 3D Tiles are an open issue.

For i3s `application/vnd.esri.i3s.json+gzip` is specified, but not registered with IANA yet.

For 3D Tiles no specific media type has been specified yet and general media types are used

[https://github.com/AnalyticalGraphicsInc/3d-tiles/tree/master/specification#file-extensions-and-mime-types] (in particular `application/json`). This should be changed as the use of the general media types is ambiguous.

The Hochschule für Technik Stuttgart has implemented a prototype of such an API as a facade on top of the 3DPS implementation from Testbed 13, which uses the New York dataset.

### 7.4.3. Other 3DPS requests

In addition to scenes, the 3DPS 1.0 standard [19] specifies additional requests that would need to be mapped, too. A detailed analysis is out-of-scope for this report, but from looking at the feature info requests we can see the benefits of the NextGen architecture that supports the different resources derived from a dataset in a single API. Where 3DPS 1.0 has to define its own requests to access feature data, in the NextGen architecture these resources are often already provided by the API building blocks specified by WFS 3.0.

`GetFeatureInfoByObjectId` in 3DPS 1.0, for example, is an operation that allows a client to retrieve information about features that are selected based on object identifiers. In the NextGen architecture this is simply the paths `/collections/{collectionId}/items/{featureId}` specified by WFS 3.0 Core.

On the other hand `GetFeatureInfoByRay` introduces a new way of spatially selecting features based on a virtual ray. In the NextGen architecture this would be implemented in an extension to the WFS 3.0 `/collections/{collectionId}/items` resource where additional query parameters specify the ray.

# Chapter 8. Recommendations

This chapter builds on the analysis in the previous chapters and derives recommendations for the most appropriate way to support the use cases within the NextGen service architecture.

## 8.1. Recommendation 1a: Specify WFS 3.0 extensions for improved fetching of feature data

This could be multiple documents or a single document with multiple conformance classes.

1. One conformance class should support CQL queries on dataset and collection resources as outlined here. The CQL should be restricted to the profile that is implemented in GeoTools and other libraries (e.g. no support for compound attributes, relationships, attributes with multiple values). Alternatively, another simple, easy to implement syntax that is straightforward to map to queries on relational databases and document stores could be implemented. The STAC JSON syntax could be an option.
2. If CQL is supported, another conformance class should add support for the richer queries including the capabilities for the "complex feature handling" use cases discussed in this document (see these examples).
3. Add a path `/items` in another conformance class to support fetching features from all the sub-collections in a dataset. A query parameter `collections` should be added to select features from a subset of the collections.
4. In another conformance class, add a `resolve` query parameter to the feature collection and feature resources as outlined here. If the CQL queries support nested structures and relationships (see item 2 above), the `resolve` parameter should supported the extended property notation, too.
5. In another conformance class, add a `properties` query parameter to the feature collection and feature resources as outlined here. Note that additional conformance classes will need to be specified for encodings how to map property names to the content. This should be done for the GeoJSON and GML encodings specified in the WFS 3.0 Core. Again, if the CQL queries support nested structures and relationships (see item 2 above), the `properties` parameter should support the extended property notation, too.

In addition, the Core should be amended to support the capability to select, by default, only features that are valid "now". This is useful for datasets where the features are versions of a real-world entity, valid for a given time period. "By default" means, without an explicit `time` parameter. The default behavior could be stated in the feature collection metadata, the default would be the current WFS 3.0 behaviour (all features that meet the explicit selection criteria).

## 8.2. Recommendation 1b: Make CQL an OGC standard on its own

If CQL would be supported by a WFS extension as a result of the recommendation 1 above, it should be extracted from the Catalogue Service standard and become a standard on its own.

The CQL standard should provide a clear and unambiguous specification of the query language and requirements for implementations. It also clarify how CQL provides API building blocks that can be reused by WFS 3.0 or other NextGen standards.

## 8.3. Recommendation 2: Migrate additional OGC web service standards to the NextGen architecture

The analysis has outlined how current 3DPS, WMTS and WMS that operate on feature data could be implemented in the NextGen architecture. Other activities in OGC Testbed 14 (see, for example deliverable D021 "Secure Resource Oriented Geospatial Data Handling Engineering Report") as well as the OGC Vector Tiles Pilot have made experiments along the same lines.

These activities should continue to validate the NextGen goal to support access to spatial data using consistent API building blocks. This is inline with the ongoing NextGen discussions in the OGC Architecture Domain Working Group (DWG) and the OGC OWS Common SWG.

## 8.4. Recommendation 3: Investigate if/how GraphQL could be used to query a dataset

GraphQL is a highly popular query mechanism to fetch data with support for complex data structures. It is relatively straightforward to use and at the same time allows highly customizable queries that return exactly the data that the client wants.

Its architecture is orthogonal to the NextGen architecture which is based on the architecture of the web. That is, support for GraphQL would be complementary to the WFS 3.0 / NextGen approach.

It is recommended to explore how GraphQL endpoints could support complex feature handling use cases in more detail than it was possible in Testbed 14. This should include prototypes.

Experiments should focus on JSON encodings.

Open questions include:

- To what extent should existing encodings like GeoJSON and CityJSON be used in GraphQL schemas?
- Would there be any value in having a general mapping from an application schema to a GraphQL query schema?
- How can GraphQL queries be extended to support spatial searches - without the need to discard the tooling?

In Testbed-14 an initial, simple experiment has been included that looked into the first of the two questions. The results seem to indicate that GraphQL schemas for querying feature data probably should not use GeoJSON/CityJSON, but tailored schemas for the specific query needs. If this conclusion is confirmed by additional, more substantial experiments, then supporting a GraphQL query endpoint for feature data should not be subject to standarization as a WFS extension. Instead guides or tutorials would be more appropriate and useful.

Or, alternatively, there could be value in specifying a general (default) mapping from an application schema to a GraphQL query schema. This could be used as a starting point, too, and be tailored towards the specific application needs of the deployment.

In addition, the third question could be a potential subject of future standarization. That is, how to consistently support spatial searches in GraphQL queries.

## 8.5. Recommendation 4: Consider the use of CityJSON for 3D city models

Due to the current popularity of JSON (over XML), WFS 3.0 implementations should not only support an XML/GML-based encoding, but also a JSON-based encoding. As GeoJSON does not meet the needs of 3D city models, CityJSON is a primary starting point. CityJSON is not yet a standard, but investigations are ongoing [https://github.com/w3c/strategy/issues/114].

## 8.6. Recommendation 5: Develop guidance for implementing additional search capabilities

Provide guidance on how implementations could support additional, more advanced query mechanisms that best fit their encoding(s) and datastore. These probably should not be promoted as OGC standards as the details may be implementation dependent and too few implementations would support the same advanced capabilities.

Examples could be:

- the use of XQuery on a document store that supports XML or a XML database;
- the use of WFS 2.0 Query / FES 2.0 Filter in an implementation that supports WFS 2.0 in addition to WFS 3.0;
- the use of (Geo)SPARQL in an implementation that uses a triple store;
- the use of a bespoke query language with a straightforward mapping to the underlying datastore (for example, a WFS 3.0 that uses a MongoDB datastore probably would support a query mechanism that directly maps to the native MongoDB queries [https://docs.mongodb.com/manual/tutorial/query-documents/]).

The general considerations on the API design discussed in chapter 7 should be taken into account when developing the guidance.

In particular for query languages that are close to the query mechanisms of the backend datastore, the security implications should be considered and servers should inspect a query thoroughly before executing it on the backend datastore (for example, the queries should be analysed for malicious content or to verify that they do not place an undesired burden on the server. It should not be forgotten that exposing rich query capabilities via a Web API is a potential security risk.

## 8.7. Recommendation 6: Validate and refine recommendations through implementation

In general, the recommendations above have not been tested yet in implementations. Validation and refinement through implementation is, however, fundamental to the WFS 3.0 and NextGen process. All recommendations should thus be validated in multiple implementations before considering them for standardisation.

## 8.8. Recommendation 7: Register media types for encodings to be used in Web APIs

The selection of the data format used in a response must support HTTP content negotiation. This requires that format can be identified using media types that should be registered with IANA. For example, media types should be registered for i3s, 3D Tiles, CityJSON and perhaps also for CityGML.

# Appendix A: Revision History

*Table 3. Revision History*

| Date | Editor | Release | Primary clauses modified | Descriptions |
|---|---|---|---|---|
| April 11, 2018 | C. Portele | 0.1 | 1 | initial version |
| May 8, 2018 | C. Portele | 0.2 | 2-8 | add document structure, initial use case template |
| May 30, 2018 | V. Coors, C. Portele | 0.3 | 5 | initial use cases added (work in progress) |
| June 22, 2018 | V. Coors, C. Portele | 0.4 | 5 | additional use cases and details added (work in progress) |
| June 28, 2018 | C. Portele | 0.5 | 5 | update use cases, unify structure and style, add bibliography |
| September 4, 2018 | C. Portele | 0.6 | 6-7 | initial versions of chapters 6 and 7 |
| October 1, 2018 | C. Portele | 0.7 | 1,6-8 | additional content plus updates based on discussions between Clemens and Volker |
| October 3, 2018 | C. Portele | 0.8 | all | update after review by Gobe Hobana |
| October 5, 2018 | C. Portele | 0.9 | title, 1, 8 | change title to include "Next Generation APIs", editorial updates |
| October 8, 2018 | C. Portele | 0.10 | 7, 8 | add GraphQL example |

| Date | Editor | Release | Primary clauses modified | Descriptions |
|---|---|---|---|---|
| October 11, 2018 | C. Portele | 0.11 | 1.3, 7.3.3 | add content to the future work section, issues #1 and #2 |
| October 16, 2018 | C. Portele | 0.12 | 8.1 | add paragraph about the time parameter, update title |
| October 22, 2018 | C. Portele | 0.13 | title, 1 | another title update |
| October 31, 2018 | C. Portele | 0.14 | 7, 8 | updates on recommendations 3 and 5, add clarification on queryables |
| November 19, 2018 | C. Portele | 0.15 | 7.4.2 | add reference to 3DPS prototype of HfT |
| December 1, 2018 | P. Rodrigues, C. Portele | 0.16 | 7.4.2 | add detail about the 3DPS prototype |
| December 13, 2018 | C. Portele | 0.17 | 5.2.5, 7.4.1 | use "tiled vector data" instead of "vector tiles" |

# Appendix B: Bibliography


1. Tandy, J., Brink, L. van den, Barnaghi, P. eds: Spatial Data on the Web Best Practices. https://www.w3.org/TR/sdw-bp/. World Wide Web Consortium, Open Geospatial Consortium (2017).
2. Energy Concept for an Environmentally Sound, Reliable and Affordable Energy Supply. http://www.bmwi.de/English/Redaktion/Pdf/energy-concept. Federal Ministry of Economics and Technology (2010).
3. Monien, D., Strzalka, A., Koukofikis, A., Coors, V., Eicker, U.: Comparison of building modelling assumptions and methods for urban scale heat demand forecasting. Future Cities and Environment. http://rdcu.be/oHtg. Springer Open Access (2017).
4. Nouvel, R., Mastrucci, A., Coors, V., Leopold, U., Eicker, U. and: Combining GIS-based statistical and engineering urban heat consumption modelling: Towards a new framework for multi-scale policy support. https://doi.org/10.1016/j.enbuild.2015.08.021. Energy and Buildings. 107, 204–212 (2015).
5. CitySim, http://www.kaemco.ch/.
6. Chen, Y., Hong, T., Piette, M.A.: Automatic generation and simulation of urban building energy models based on city datasets for city-scale building retrofit analysis. https://doi.org/10.1016/j.apenergy.2017.07.128. Applied Energy. 205, 323–335 (2017).
7. Coors, V. ed: OGC Testbed 13 – 3D Tiles and I3S Interoperability and Performance Engineering Report. http://docs.opengeospatial.org/per/17-046.html. Open Geospatial Consortium (2018).
8. ShapeChange - Processing application schemas for geographic information, https://shapechange.net/.
9. SpatioTemporal Asset Catalog specification, https://github.com/radiantearth/stac-spec/.
10. Vretanos, P. ed: OGC Filter Encoding 2.0 Encoding Standard. https://docs.opengeospatial.org/is/09-026r2/09-026r2.html. Open Geospatial Consortium (2014).
11. Vretanos, P. ed: OGC Web Feature Service 2.0 Interface Standard. https://docs.opengeospatial.org/is/09-025r2/09-025r2.html. Open Geospatial Consortium (2014).
12. Nebert, D., Whiteside, A., Vretanos, P. eds: OpenGIS Catalogue Services Specification. http://portal.opengeospatial.org/files/?artifact_id=20555. Open Geospatial Consortium (2007).
13. Falcor website, https://netflix.github.io/falcor/.
14. What is Falcor?, https://netflix.github.io/falcor/starter/what-is-falcor.html.
15. GraphQL website, https://graphql.org/.
16. Knyga, O., Kolesnikov, M., Guliaev, S., Eremin, V., Hayat, S., Dmytriienko, D.: GraphQL - Everything You Need to Know. https://medium.com/@weblab_tech/graphql-everything-you-need-to-know-58756ff253d8. (2018).
17. Simmons, S.: OGC advances the Web Feature Service standard through a public hackathon. http://www.opengeospatial.org/blog/2764. (2018).
18. Holmes, C.: Progress on SpatioTemporal Asset Catalogs in Ft. Collins. https://medium.com/radiant-earth-insights/progress-on-spatiotemporal-asset-catalogs-in-ft-collins-6298f195bfb2. (2018).

19. Hagedorn, B., Thum, S., Reitz, T., Coors, V., Gutbell, R. eds: OGC 3D Portrayal Service 1.0. http://docs.opengeospatial.org/is/15-001r4/15-001r4.html. Open Geospatial Consortium (2017).
20. Reed, C., Belayneh, T. eds: OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package Format Specification. http://docs.opengeospatial.org/cs/17-014r5/17-014r5.html. Open Geospatial Consortium (2017).
21. Cozzi, P., Lilley, S., Getz, G. eds: 3D Tiles Specification 1.0. https://portal.opengeospatial.org/files/?artifact_id=79137&version=3. Open Geospatial Consortium (2018).