OGC Testbed-15

Delta Updates Engineering Report

Table of Contents

Publication Date: 2019-12-17

Approval Date: 2019-11-22

Submission Date: 2019-10-29

Reference number of this document: OGC 19-012r1

Reference URL for this document: http://www.opengis.net/doc/PER/t15-D005

Category: OGC Public Engineering Report

Editor: Benjamin Pross

Title: OGC Testbed-15: Delta Updates Engineering Report

OGC Public Engineering Report

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. Subject

This OGC Testbed 15 Engineering Report (ER) documents the design of a service architecture that allows the delivery of prioritized updates of features to a client, possibly acting in a DDIL (Denied, Degraded, Intermitted or Limited Bandwidth) environment. Two different technical scenarios were investigated and tested:

- The enhancement of Web Feature Service (WFS) instances to support updates on features sets.
- Utilizing a Web Processing Service (WPS) instance to access features, without the need to modify the downstream data service.

Chapter 2. Executive Summary

Dissemination of GEOINT data in a Denied, Degraded, Intermittent and Limited (DDIL) Bandwidth environment is a challenging problem. By not serving the entire dataset, but only the changes (delta updates) and also considering priority was identified as a valid approach to this problem.

The key research question was then how to implement a reliable and secure delta update mechanism using OGC next generation Web Services such as OGC API – Features and WPS/OGC API – Processes.

This ER documents how prioritized delta updates can be served using a transactional extension to the OGC API – Features and the WPS standard/OGC API – Processes in front of WFS instances. Both approaches use the same algorithm to keep track of the changes to the dataset.

Implementation details are given about the server and client components developed in the Delta Updates thread during OGC Testbed-15.

Topics for future work are:

- The handling of delta updates in simulated DDIL environments.
- Context-based prioritization, for example using a mobile client that only needs high priority updates.
- Investigate a common base for delta updates in OGC APIs

2.1. Document contributor contact points

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

Contacts

Name	Organization	Role
Benjamin Pross	52°North GmbH	Editor
Peter Vretanos	CubeWerx	Contributor
Eve Ousby	Helyx Secure Information Systems Ltd	Contributor

2.2. 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 3. References

The following normative documents are referenced in this document.

- OGC: OGC 17-069r3, OGC API - Features - Part 1: Core (2019) [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]
- OGC: OGC 12-128r14, OGC GeoPackage Encoding Standard, Version 1.2.0 (2018) [https://www.geopackage.org/spec120/]
- OGC: OGC 09-025r2, OGC Web Feature Service 2.0 Interface Standard – With Corrigendum (2014) [http://docs.opengeospatial.org/is/09-025r2/09-025r2.html]

Chapter 4. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **Delta Update**

  Update that only requires the system to download the new changes and not the whole database.

- **Changeset**

  Set of changed items. Changeset is a synonym for delta updates as are incremental updates and change only updates (COU). The term Changeset is used in OGC Testbed-15: Open Portrayal Framework Engineering Report (OGC 19-018) and OGC Testbed-15: Images and ChangesSet API Draft Specification (OGC 19-070).

- **Checkpoint**

  Point in time when updates were requested.

4.1. Abbreviated terms

- DDIL Denied, Degraded, Intermitted or Limited Bandwidth
- GEOINT Geospatial Intelligence
- TB15 OGC Testbed-15
- TIE Technology Integration Experiment
- UUID universally unique identifier
- WPS Web Processing Service
- WFS Web Feature Service
- WFS-T Transactional Web Feature Service

Chapter 5. Overview

Section 6 introduces the delta updates algorithm.

Section 7 presents how the delta updates algorithm was implemented using the new OGC API – Features Part 1: Core Standard.

Section 8 describes how the delta updates algorithm was implemented on the server-side.

Section 9 presents the delta updates client implementation.

Section 10 provides the conclusions and gives an overview on future work.

Annex A provides JSON schemas.

Chapter 6. Delta Updates Algorithm

A delta update is defined as an update that only requires the system to download the new changes (the "delta") and not the entire database or data store.

The following is a high level description of the algorithm used by the CubeWerx [http://www.cubewerx.com/] server to determine which features have been modified between two checkpoints (i.e. the delta updates). The approach uses a database to keep track of the delta updates. The approach is independent of the service interface.

6.1. Structures

This section describes the table structure used by the delta updates algorithm. The delta updates algorithm involves the use of 2 tables; an AUDIT table and a CHECKPOINT table.

The AUDIT table contains all updates. The schema is as follows:

Table 1. AUDIT table schema

Column Name	Data Type	Description
SEQ	*integer*	*Sequence, increased by one with every update,* **primary key**
TXID	*string*	*Transaction ID, e.g. handle attribute in WFS transaction requests*
TIMESTAMP	*dateTime*	*Timestamp of the update*
FEATURE_COLLECTION_ID	*string*	*ID of the feature collection*
FEATURE_ID	*string*	*ID of the specific feature*
OPERATION	*string*	*Update operation (e.g. insert, update, delete)*
PRIORITY	*string*	*Priority of the update (low, medium, high)*

The CHECKPOINT table contains information about the updates between two checkpoints. The schema is as follows:

Table 2. CHECKPOINT table schema

Column Name	Data Type	Description
CHECKPOINT	*string*	*The checkpoint,* **primary key**
FEATURE_COLLECTION_ID	*string*	*ID of the feature collection*
SEQ	*integer*	*Sequence,* **foreign key relation to AUDIT.SEQ**

6.2. Algorithm

This section presents the algorithm used to detect delta updates by way of an example of a hypothetical feature collection named BUILDINGS.

All updates are stored in the AUDIT table. The delta updates can be requested with the arguments described in the following table:

Table 3. Delta updates arguments

Arguments	Response
No arguments	*All updates are returned*
Checkpoint	*Updates since this checkpoint*
Priority	*All updates with this priority*
Checkpoint and Priority	*Updates since this checkpoint with this priority*

In the following, the algorithm is explained using actions and the resulting states of the AUDIT and CHECKPOINT table.

```
ACTION 1: INSERT 2 features in the BUILDING collection (high priority update)
```

Table 4. AUDIT table after action 1

SEQ	TXID	TIMESTAMP	FEATURE_COLLECTION_ID	FEATURE_ID	OPERATION	PRIORITY
1	*1*	*T0*	*BUILDINGS*	*10*	*INSERT*	*high*
2	*1*	*T1*	*BUILDINGS*	*11*	*INSERT*	*high*

```
ACTION 2: INSERT 1 feature into the BUILDINGS collection (high priority) and UPDATE
another feature (medium priority)
```

Table 5. AUDIT table after action 2

SEQ	TXID	TIMESTAMP	FEATURE_COLLECTION_ID	FEATURE_ID	OPERATION	PRIORITY
1	*1*	*T0*	*BUILDINGS*	*10*	*INSERT*	*high*
2	*1*	*T1*	*BUILDINGS*	*11*	*INSERT*	*high*
3	*2*	*T2*	*BUILDINGS*	*27*	*INSERT*	*high*
4	*2*	*T3*	*BUILDINGS*	*11*	*UPDATE*	*medium*

```
ACTION 3: First delta update retrieval
```

Table 6. CHECKPOINT table after action 3

FEATURE_COLLECTION_ID	CHECKPOINT	SEQ
BUILDINGS	*812b167d-7c6e-489d-a39f-4e00b6aeb5ab*	*4*

Since this is the first request for delta updates there is no checkpoint specified and the server would respond with all changes applied to the BUILDINGS feature collection since it was created (or since audit tracking was started). The checkpoint is returned to the requester along with the updates. The requester would need to keep track of the checkpoint value ("812b167d-7c6e-489d-a39f-4e00b6aed5ab" in this case, which is a universally unique identifier (UUID)).

```
ACTION 4: INSERT a feature (low priority)
```

Table 7. AUDIT table after action 4

SEQ	TXID	TIMESTAMP	FEATURE_COLLECTION_ID	FEATURE_ID	OPERATION	PRIORITY
1	*1*	*T0*	*BUILDINGS*	*10*	*INSERT*	*high*
2	*1*	*T1*	*BUILDINGS*	*11*	*INSERT*	*high*
3	*2*	*T2*	*BUILDINGS*	*27*	*INSERT*	*high*
4	*2*	*T3*	*BUILDINGS*	*11*	*UPDATE*	*medium*
5	*3*	*T4*	*BUILDINGS*	*32*	*INSERT*	*low*

```
ACTION 5: DELETE a feature (medium priority)
```

Table 8. AUDIT table after action 5

SEQ	TXID	TIMESTAMP	FEATURE_COLLECTION_ID	FEATURE_ID	OPERATION	PRIORITY
1	*1*	*T0*	*BUILDINGS*	*10*	*INSERT*	*high*
2	*1*	*T1*	*BUILDINGS*	*11*	*INSERT*	*high*
3	*2*	*T2*	*BUILDINGS*	*27*	*INSERT*	*high*
4	*2*	*T3*	*BUILDINGS*	*11*	*UPDATE*	*medium*
5	*3*	*T4*	*BUILDINGS*	*32*	*INSERT*	*low*
6	*4*	*T5*	*BUILDINGS*	*11*	*DELETE*	*medium*

```
ACTION 6: Retrieve delta update since checkpoint 812b167d-7c6e-489d-a39f-4e00b6aeb5ab
```

Table 9. CHECKPOINT table after action 6

FEATURE_COLLECTION_ID	CHECKPOINT	SEQ
BUILDINGS	*812b167d-7c6e-489d-a39f-4e00b6aeb5ab*	*4*
BUILDINGS	*ac1cfdea-6222-497f-8cee-cd8a381b8c62*	*6*

Since the last checkpoint, one feature has been inserted and one feature has been deleted; this is reflected in the changeset output.

Chapter 7. Delta Updates for OGC API – Features

This section describes how the delta updates algorithm was implemented using the OGC API – Features Part 1: Core Standard.

7.1. Storing updates

Updates are stored in an implementation of the OGC API – Features standard using transactions. Transactions for OGC API – Features can be classified as simple transactions and complex transactions. Simple transactions are modifications that affect a single feature in a single collection. Complex transactions are modifications that affect multiple features perhaps across multiple collections and they can have batch or atomic semantics. An extension for simple transactions for the OGC API – Features is currently in draft state.

This section briefly outlines:

- How both simple transactions work using the standard POST, PUT, PATCH and DELETE methods.
- Extensions to both approaches for transactions for the delta updates thread [1: a thread aggregates a number of tasks] of TB15.

In the discussion that follows, the hypothetical collection named BUILDINGS will be used for illustration purposes.

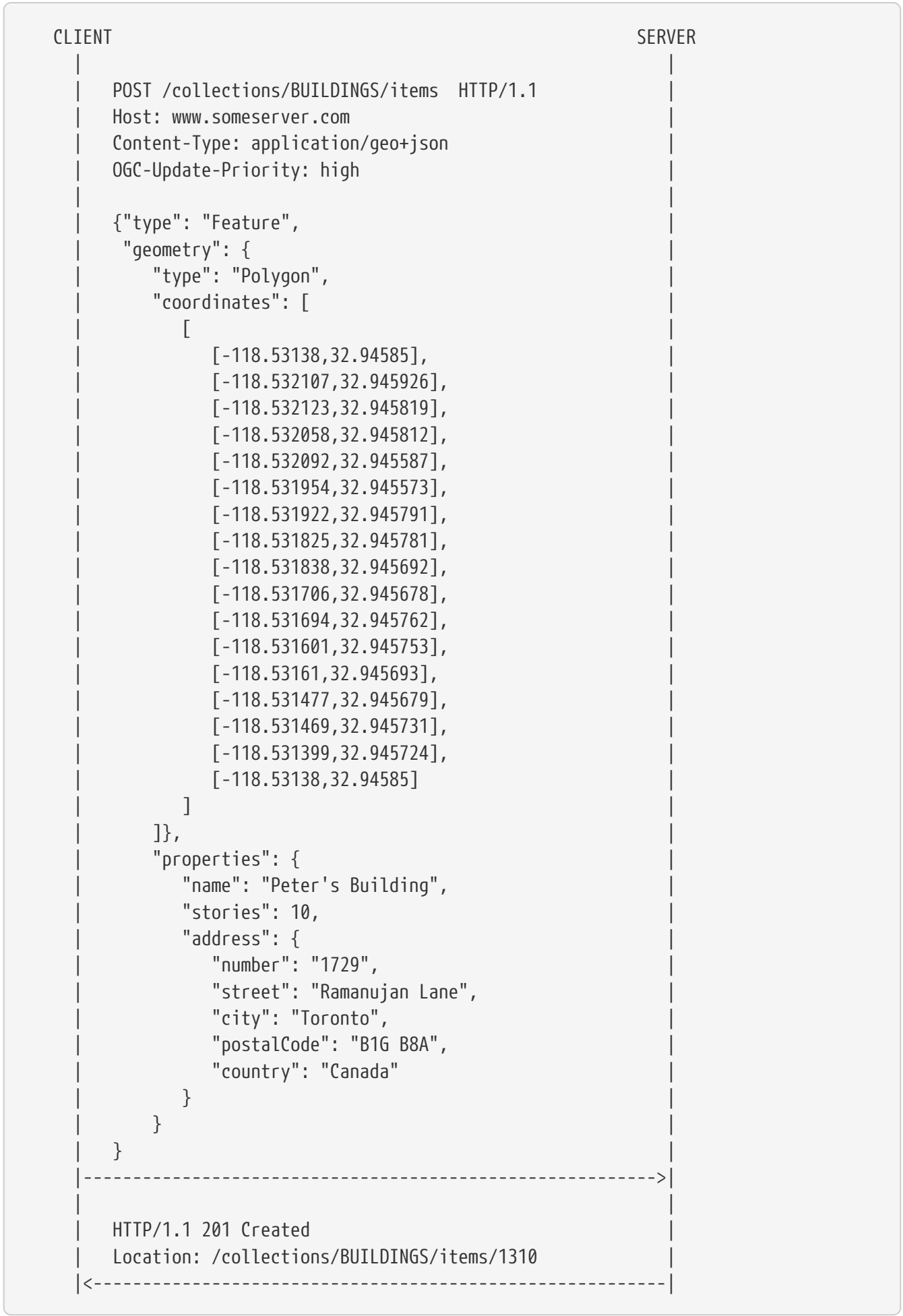
7.2. Simple Transactions

Simple transactions use the HTTP methods POST, PUT, PATCH and DELETE to create, replace, modify and remove features from a collection. The relevant OGC API – Feature resource paths are:

```
/collections/{collectionId}/items
/collections/{collectionId}/items/{featureId}
```

7.2.1. Creating a new feature

The following sequence diagram illustrates how to add a new feature to a collection:



The body of the request contains a representation of the feature to be created (in this example the

representation is GeoJSON). The Content-Type header contains the MIME type of the body of the request (GeoJSON in this case). The server responds with a HTTP code 201 indicating that the new feature was created. The server's response also includes a "Location" header to let the client know the URL of the newly created feature. The OGC-Update-Priority header is used by the delta update thread to tag the operation with a priority

- In this case, this insert is tagged as a high priority update.
- For OGC Testbed-15 the priority vocabulary is: high, medium and low.

7.2.2. Replacing an existing feature

The following sequence diagram illustrates how an existing feature may be replaced with an updated feature. In this case the footprint of the building has been changed.

```
CLIENT                                                    SERVER
  |                                                          |
  |   PUT /collections/BUILDINGS/items/1310  HTTP/1.1        |
  |   Host: www.someserver.com                               |
  |   Content-Type: application/geo+json                     |
  |   OGC-Update-Priority: medium                            |
  |                                                          |
  |   {                                                      |
  |      "type": "Feature",                                  |
  |      "geometry": {                                       |
  |         "type": "Polygon",                               |
  |         "coordinates": [                                 |
  |            [                                             |
  |               [-118.541296,32.9395],                     |
  |               [-118.541369,32.939566],                   |
  |               [-118.541436,32.939513],                   |
  |               [-118.541409,32.939489],                   |
  |               [-118.541385,32.939508],                   |
  |               [-118.541338,32.939467],                   |
  |               [-118.541296,32.9395]                      |
  |            ]                                             |
  |         ]                                                |
  |      },                                                  |
  |      "properties": {                                     |
  |         "name": "Peter's Building",                      |
  |         "stories": 10,                                   |
  |         "address": {                                     |
  |            "number": "1729",                             |
  |            "street": "Ramanujan Lane",                   |
  |            "city": "Toronto",                            |
  |            "postalCode": "B1G B8A",                      |
  |            "country": "Canada"                           |
  |         }                                                |
  |      }                                                   |
  |   }                                                      |
  |--------------------------------------------------------->|
  |                                                          |
  |   HTTP/1.1 200 OK                                        |
  |<---------------------------------------------------------|
```

In the above example, the PUT method is applied to the URL of the feature to be replaced. The body of the request contains a representation of the new or updated feature. In this case the representation is GeoJSON. The server's response is HTTP code 200 (OK).

7.2.3. Partially updating an existing feature

The following sequence diagram illustrates how to partially update an existing feature without having to replace the entire feature.


CLIENT
SERVER
PATCH /collections/BUILDINGS/items/1310 HTTP/1.1
Host: www.someserver.com
Content-Type: application/????+json
Accept: application/geo+json
OGC-Update-Priority: medium
{
"add": [
{
"name": "status",
"value": "Under renovation"
}
],
"modify": [
{
"name": "stories",
"value" 73
}
]
}
HTTP/1.1 200 OK
Content-Type: application/geo+json
Location: /collections/BUILDINGS/items/1310
{
"type": "Feature",
"geometry": {
"type": "Polygon",
"coordinates": [
[
[-118.541296,32.9395],
[-118.541369,32.939566],
[-118.541436,32.939513],
[-118.541409,32.939489],
[-118.541385,32.939508],
[-118.541338,32.939467],
[-118.541296,32.9395]
]
]
},
"properties": {
"name": "Peter's Building",
"stories": 73,
"status": "Under renovation"
"address": {
"number": "1729",
"street": "Ramanujan Lane",

```
|            "city": "Toronto",                                  |
|            "postalCode": "B1G B8A",                            |
|            "country": "Canada"                                 |
|         }                                                      |
|      }                                                         |
|   }                                                            |
|<---------------------------------------------------------------|
```

In the above example, the PATCH method is applied to the URL of the feature. The body of the request contains a JSON document that contains instructions about how the feature should be modified.

- The instructions indicate that the "status" property should be added to the feature.
- The instructions indicate that the value of the "stories" property should be changed from its existing value (i.e. 10) to 73.

The server's response is HTTP code 200 (OK). The response body contains the new state of the feature in some representation based on the value of the Accept header (GeoJSON in this case).

The schema defining the body of the PATCH request can be found in annex JSON Schema Listings

NOTE	The add and remove actions for the PATCH result in changes of the data schema which in some cases (such as a system backed by an RDBMS) could be difficult to implement.

7.2.4. Delete a feature

The following sequence diagram illustrates how a feature may be deleted or removed from a collection.

```
CLIENT                                                    SERVER
  |                                                          |
  |   DELETE /collections/BUILDINGS/items/1310  HTTP/1.1     |
  |   Host: www.someserver.com                               |
  |   OGC-Update-Priority: low                               |
  +--------------------------------------------------------->|
  |                                                          |
  |   HTTP/1.1 200 OK                                        |
  +<---------------------------------------------------------|
```

In the above example, the DELETE method is applied to the URL of the feature. The server's response is HTTP code 200 (OK).

7.3. Getting Updates

The access path for features in the OGC API – Features Standard is the path:

```
/collections/{collectionId}/items
```

The result of accessing this path is to retrieve a subset of features from the specified collection (identified by the {collectionId} substitution variable).

A delta update is the subset of features that have changed between two checkpoints and as such is itself a subset of features from the specified collection. So, it is proposed that an access path similar to that used to fetch features could be used to access delta updates. Specifically:

```
/collections/{collectionId}/changesets/{checkpointId}
```

The {checkpointId} parameter is optional. Without it, the response to accessing the delta update path would be all changes since auditing began. Filters such as bounding box, time, and so forth may be applied to the path to request a specific subset inserted or updated features.

The following additional parameters may also be used with the delta updates path:

Table 10. Delta Updates path parameters

Parameter Name	**Value**	**Description**
resultType	*one of: summary,* **full**	*The value "full" causes the server to return the complete changeset. This is the default value.* *The value "summary" causes the server to return a summary of the changes.*
priority	*one of: low, medium or high*	*One or more priority labels used to filter a delta update response.*

The response to a delta update request is called a changeset. The JSON schema for the changeset can be found in annex JSON Schema Listings.

7.3.1. Examples

The following example requests a summary of the delta updates since the example checkpoint "cp143756":

```
/collections/BUILDINGS/changesets/cp143756?resultType=summary
```

Example response:

```
{
   "checkPoint": "cp143756",
   "summaryOfChangedItems": [
      { "priority": "high", "count": 2 },
      { "priority": "medium", "count": 4 },
      { "priority": "low", "count": 67 }
   ]
}
```

The following example requests the medium and high priority delta updates since the example checkpoint "cp143756":

```
/collections/BUILDINGS/changesets/cp143756?resultType=full&priority=high,medium
```

Example response:

```
{
   "checkPoint": "cp143756",
   "summaryOfChangedItems": [
      { "priority": "high", "count": 2 },
      { "priority": "medium", "count": 4 },
      { "priority": "low", "count": 67 }
   ],
   "numberOfReturnedItems": 6,
   "changedItems": [
      {
         "priority": "high",
          "items": [
            { ... BUILDING feature as GeoJSON ...},
            { ... BUILDING feautre as GeoJSON ...}
         ]
      },
      {
         "priority": "medium",
         "items": [
            { ... BUILDING feature as GeoJSON ...},
            { ... BUILDING feautre as GeoJSON ...}
         ]
      }
   ],
   "deletedItems": [
      {
         "priority": "medium",
         "items": [
            "/collections/BUILDINGS/items/F4674",
            "/collections/BUILDINGS/items/F37465819"
         ]
      }
   ]
}
```

7.4. HTTP conditional requests

One alternative approach for the proposed delta updates API considered during the design phase of Testbed 15 was the use of HTTP conditional requests (see https://tools.ietf.org/html/rfc7232). Conditional requests are HTTP requests (see RFC-7231) that include one or more header fields indicating a precondition to be tested before applying the method semantics (e.g. GET) to the target resource.

HTTP conditional requests were not used as the delta updates API because HTTP conditional requests are meant to operate on an entire resource. In the case of delta updates, what is being requested is the set of features that have been added, modified or removed from the collection of features between two checkpoints and so, the resource in question is the entire feature collection. This implies that each time a HTTP conditional request satisfied its preconditions, the server would

need to return the entire collection leaving it to the client to sort out which features have changed in the collection and how they have been changed. This approach would only be practical for very small feature collections and so the use of HTTP conditional requests was abandoned early on in favor of the approach described in this engineering report which supports more fine grained access to the changes made to a feature collections (i.e. the resource).

Chapter 8. Server implementations

8.1. Delta Updates WPS

This section describes how the delta updates algorithm was implemented using the OGC Web Processing Service (WPS). The WPS serves as facade for a standard WFS implementation. As there are many implementations of WFS that do not yet implement the OGC API – Features, the Delta Updates WPS facade could be used to enable delta update functionality for existing WFS instances. For Testbed-15, GeoServer was used for the downstream WFS. The Delta Updates WPS implementation was implemented using the 52°North javaPS framework. This approach implemented the current draft of the OGC API – Processes [2: https://rawcdn.githack.com/opengeospatial/wps-rest-binding/master/docs/18-062.html]. The OGC API – Processes is the next version of the WPS [1], focusing on a simple RESTful core specified as reusable OpenAPI [3: http://openapis.org/] components with responses in JSON and HTML.

The Delta Updates implementation based on the OGC API – Features uses simple transactions, as HTTP methods like POST, PATCH, DELETE can be executed directly on collection or feature paths. This is not feasible when using a WPS facade, as there is no direct mapping between processes and the collection or feature paths. Instead, two processes are used to store and get delta updates from the downstream WFS:

- org.n52.project.tb15.du.StoreUpdatesProcess
- org.n52.project.tb15.du.GetUpdatesProcess

Since the collection and feature information cannot be extracted from the path, the Delta Updates WPS uses the complex transaction approach described in the following section.

8.1.1. Complex Transactions

Complex transactions may, in a single operation, act on multiple features possibly across multiple collections. Complex transactions may have atomic or batch semantics.

- Typically atomic transactions are required if you need to make a set of changes to a number of features (possibly across multiple collections) and those changes have to all succeed or the entire operation fails and everything is rolled back (so as not to leave the server's datastore in an inconsistent state).
- Typically, batch transactions are required when you need to perform a large number of actions such as a batch insert of thousands of features.

The Delta Updates WPS only supports atomic transactions.

Document Approach

The document approach is similar to the approach specified using the WFS 2.X standard.

The document contains a complete description of the transaction. The transaction document is composed of the series of insert, update, replace or delete actions. Features from one or more collections may be affected by the actions in a transaction.

The following sequence diagram shows an INSERT ad UPDATE action in an atomic transaction:

```
CLIENT                                                  SERVER
   |                                                       |
   |   POST /transactions   HTTP/1.1                       |
   |   Host: www.someserver.com                            |
   |   Content-Type: application/ogc-tx+json               |
   |   Accept: application/json                            |
   |                                                       |
   |   {                                                   |
   |      "semantic": "atomic",                            |
   |      "transaction": [                                 |
   |         {                                             |
   |            "action": "insert",                        |
   |            "collection": "/collections/CABLE_SEGMENTS",|
   |            "directives": {                            |
   |               "comment": "add a new fibre segment",   |
   |               "priority": "high"                      |
   |            },                                         |
   |            "item": {                                  |
   |               "type": "Feature",                      |
   |               "geometry": {                           |
   |                  "type": "LineString",                |
   |                  "coordinates":                       |
   |                     [                                 |
   |                        [43.735915,-79.298053],        |
   |                        [43.735839,-79.298026]         |
   |                     ]                                 |
   |               },                                      |
   |               "properties": {                         |
   |                  "segmentId": "SEG237",               |
   |                  "manufacturer": "Corning",           |
   |                  "all-dielectric": "yes",             |
   |                  "armored": "yes",                    |
   |                  "fibre_count": 12                    |
   |                  "fibre_type": "OM3 LAZER-OPTIMIZED 50µ", |
   |                  "rodent_protected: "yes",            |
   |                  "USDPSC_code": 26126170              |
   |               }                                       |
   |            }                                          |
   |         },                                            |
   |         {                                             |
   |            "action": "update",                        |
   |            "collection": "/collections/"POLES",       |
   |            "directives": {                            |
   |               "comment": "move pole",                 |
   |               "priority": "high"                      |
   |            },                                         |
   |            "properties": {                            |
   |               "modify" [                              |
```

The JSON schema for the transaction and the transaction response can be found in annex JSON Schema Listings

A transaction is composed of an array of insert, replace, update and/or delete actions. The semantic key is used to indicate whether this transaction should be executed using atomic or batch semantics. The directives section is used to include additional metadata or directions for executing the transaction.

- The id directive is used to assign a local identifier to the action for the purpose of error

reporting e.g. if id="INSERT1" then a more meaningful error message such as "Action INSERT1 failed" in an exception report

- The comment key may be used to assign a human-readable comment about the action.
- The priority key is used to assign a priority to the action (i.e. one of high, medium or low).
  - This key allows a specific priority to be assigned to each action.
  - A client can also set the OGC-Update-Priority header to set the priority for all actions in the transaction (but can be locally overridden using the priority key).
- The directives section is extensible (i.e. other keys may be added as required but their meaning is not described in this document).

The response body contains a summary of the transaction.

- The response body indicates the number of insert, replace, update and/or delete actions were performed.
- The response body contains a set of arrays containing the identifiers of each feature affected by the transaction.
- In the case of batch semantics, the response body may also contain an array of exception reports for each action that failed.
  - Including an identifier for each action (via the directive object) so that each exception can be correlated to the action that failed is recommended.

WPS Implementation

For the WPS implementation, the following datasource was used: https://github.com/microsoft/USBuildingFootprints

The following image shows the building footprints for Washington, D.C.

Figure 1. Building footprints for Washington, D.C.

The footprint data was loaded into a PostGIS database, which was added as a layer to the GeoServer WFS and enabled for transactions.

As the WPS needs to keep track of the updates, the transactions to the WFS needed to be handled by a process as well.

The process for storing updates has the following parameters:

Table 11. Store udates process parameters

Parameter Name	**Value**	**Description**
transaction	*transaction.json* *WFS transaction request*	*Input, the update actions encoded in JSON or XML.*
priority	*one of: low, medium or high*	*Input, one or more priority labels for this delta update.*

Parameter Name	Value	Description
update-information	*changeset.json* *WFS transaction response*	*The update information encoded in JSON or XML.*

The following diagram shows the sequence for storing new updates:

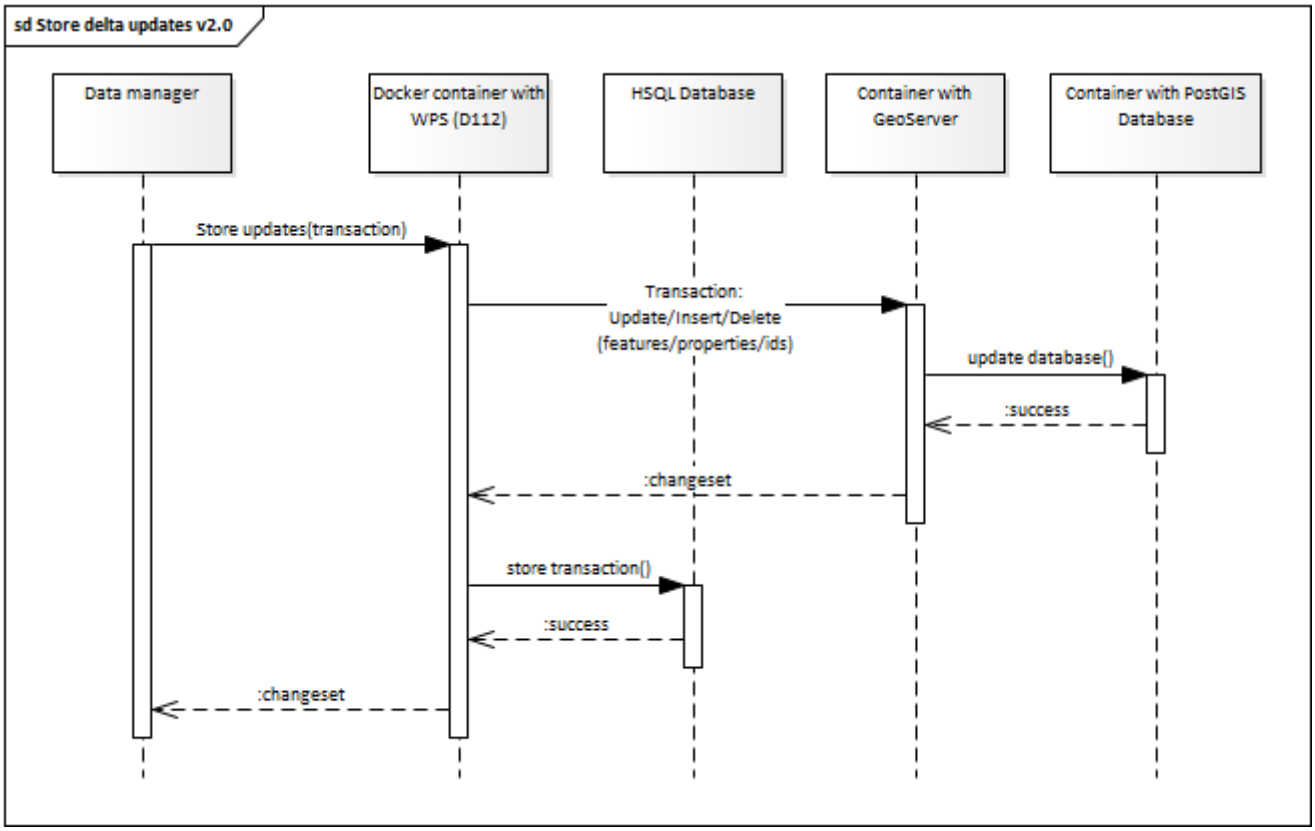


Figure 2. The sequence for storing new updates

A data manager client sends an execute request to the StoreUpdates process endpoint. The request needs to include transaction information encoded either in XML or JSON.

The process forwards the transaction information to the WFS. In case the transaction information is encoded in JSON, the process transforms it to a Transactional Web Feature Service (WFS-T XML) transaction request.

The WFS returns a changeset document. Relevant information (e.g. feature ids) is extracted from this document and used to store the transaction in the internal WPS database.

The changeset is returned to the client.

For getting updates, the client needs to specify the feature type name, the priority and the result type. Optionally, a checkpoint id can be specified. Initially, the client does not have a checkpoint id.

Table 12. Get updates process parameters

Parameter Name	Value	Description
typename	*String*	*Input, the typename of the features.*
priority	*one of: low, medium or high*	*One or more priority labels used to filter a delta update response.*
checkpoint	*String*	*Input, the checkpoint id. This is optional.*
resultType	*one of: summary, full*	*The value "full" causes the server to return the complete changeset.*
changeset	*changeset.json*	*The delta updates.*

The sequence for an initial execution of the GetUpdates process looks like the following:

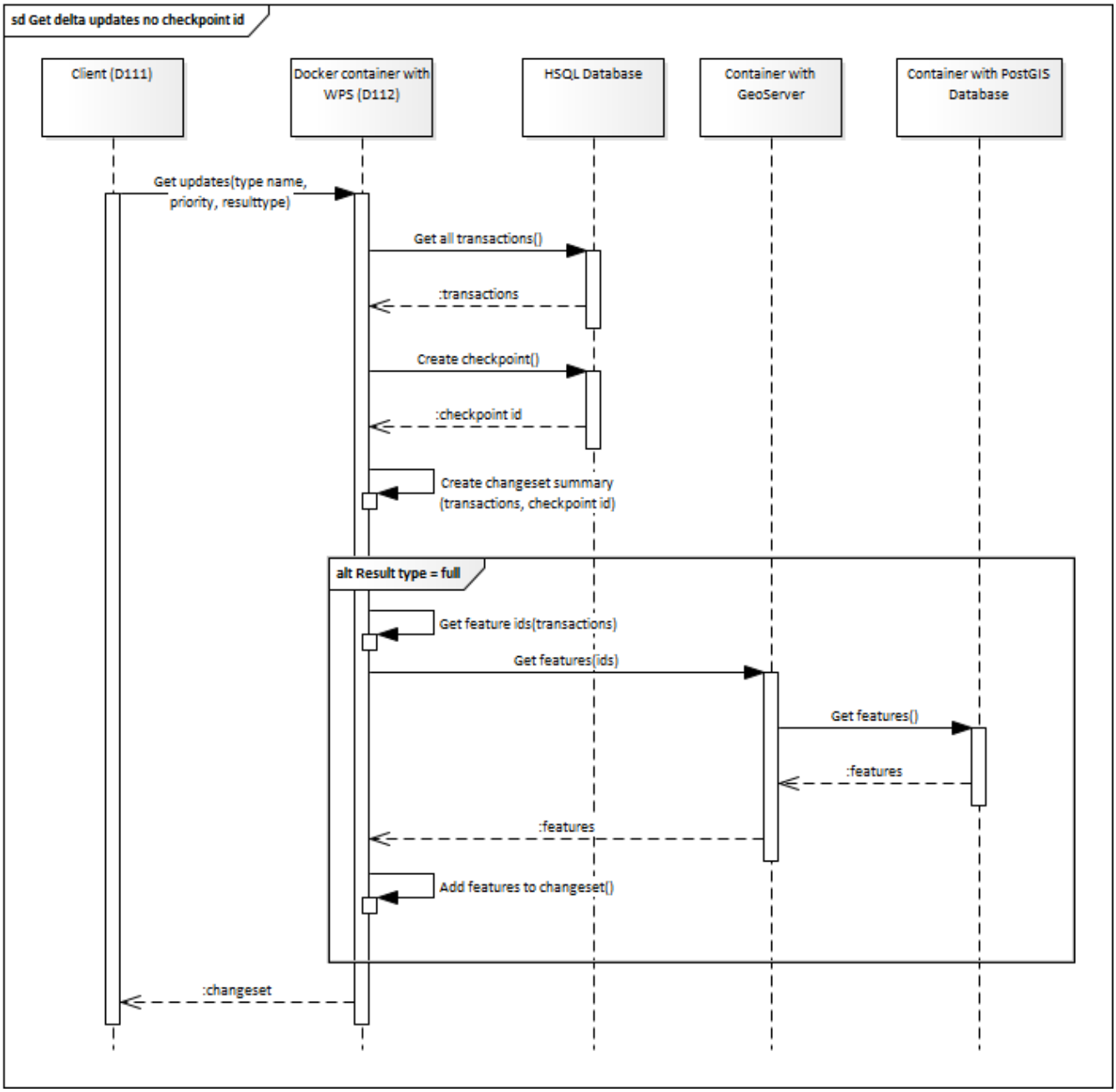


Figure 3. The sequence for an inital execution of the GetUpdates process

The process gets all transactions from the internal database. Now, a checkpoint is created and the id, together with the transaction information (transaction type [Update/Insert/Delete], feature ids,

feature type name) is stored in a changeset summary.

If the result type "full" is specified in the execute request, the ids of the transactions are used to get updated/inserted features from the WFS. The features are added to the changeset encoded in GeoJSON.

The sequence for an execution of the GetUpdates process with checkpoint id looks like the following:

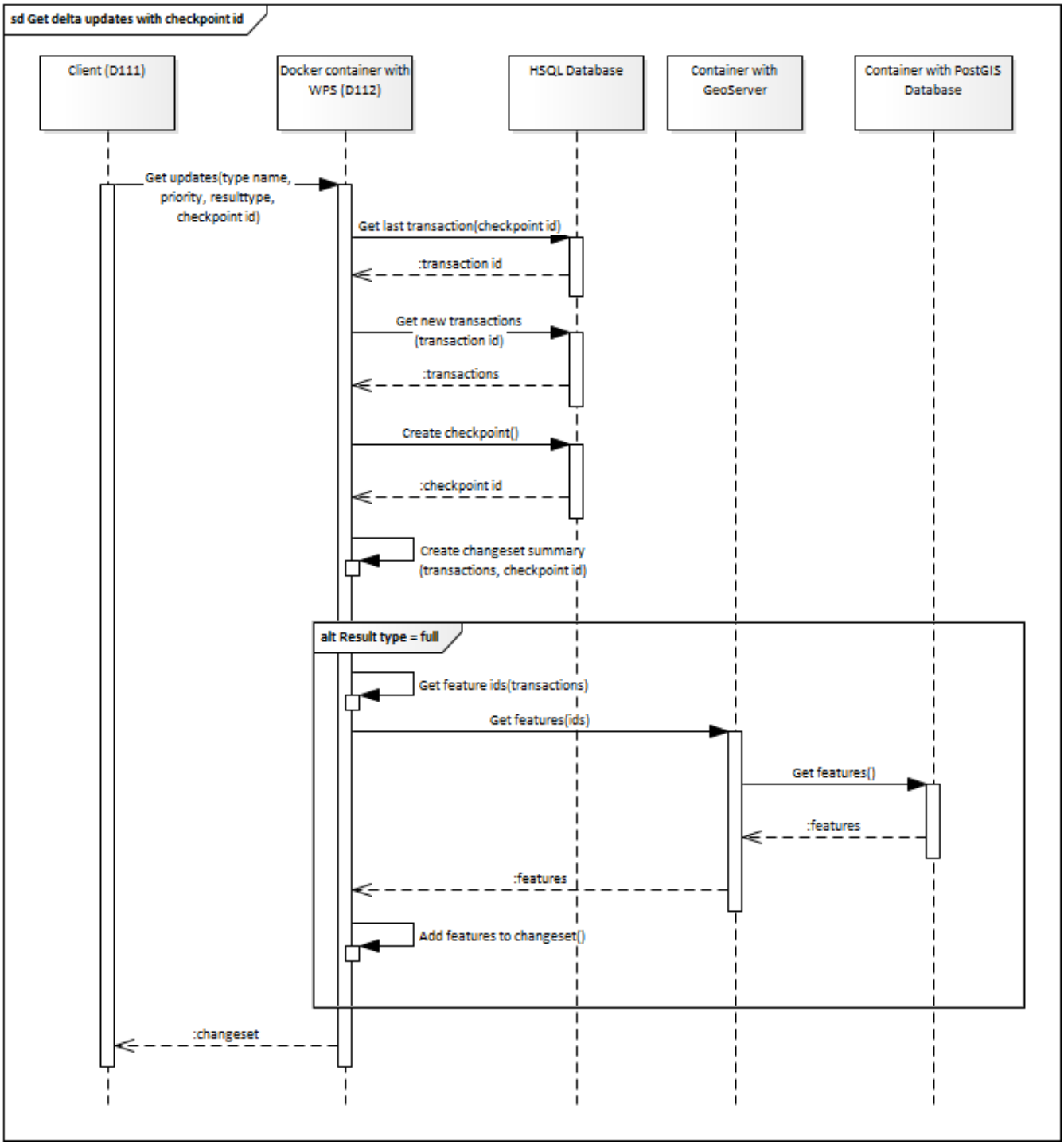


Figure 4. sequence for an execution of the GetUpdates process with checkpoint id

In contrast to the initial GetUpdates request, only the transactions that happened since the checkpoint are returned.

8.2. Delta Updates WFS (CubeWerx)

8.2.1. Implementation

This section describes CubeWerx's implementation of the delta update algorithm.

The base feature server used for the test bed is CubeWerx's WFS which is a component of an integrated geo-web server named `cubeserv`. The feature server component of `cubeserv` implements every version of the WFS standard starting from version 1.0 and extending all the way to the latest OGC API - Features - Part 1: Core standard (OAPIF).

Cubeserv, and consequently CubeWerx's feature server, is implemented in C and uses Oracle's OCI library (see https://docs.oracle.com/en/database/oracle/oracle-database/18/lnoci/index.html) to access the Oracle backend database.

The delta update algorithm was added to `cubeserv` as an extension and is accessible through all feature server interfaces:

- As a vendor extension operation (named `Sync`) in WFS request versions less than WFS version 2.X
- Via the resource paths described in clause 7 for the latest version of the WFS standard.

The specific URL template for getting delta updates via the OAPIF interface is:

/{checkpointId} [http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/{collectionsId}/changesets]

NOTE	Long lines in the examples in this clause have been wrapped to fit within the boundaries of the sequence diagram.

8.2.2. Landing Page

The landing page for the CubeWerx delta updates-enabled feature service for OGC Testbed 15 can be found at this URL:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints?f=json

Data

For the delta updates thread, the entire Microsoft US Building footprints data was loaded into an Oracle database (version 18c). The source data files can be found here: https://github.com/microsoft/USBuildingFootprints.

However, after some testing it was found that working with a data set this large was inconvenient and several issues presented themselves. The primary issues encountered were related to time and size. The very first delta update request would return entire collection of features because at Time 0 every feature in the collection is considered a delta from the empty set. Fetching the entire 120+ million features through the WFS API would take many hours. The size of the data set also meant that features would need to be fetched in batches and this caused resource problems on the server once 40 or 50 million footprints had been fetched. As a result of these issues, the participants

decided that a smaller subset of the data around Washington DC would be used instead.

The collection endpoints for both the entire set of footprints and the DC subset can be found here:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collections/DC_Building_Footprints?f=json

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collections/DC_Building_Footprints?f=json

8.2.3. Transactions

Introduction

The issue of how transactions are actually performed is somewhat orthogonal to the issue of delta updates. The only requirement for delta updates is that transactions are somehow performed on a server in order to generate changes that may then be reported using the delta update mechanism.

The Simple Transactions and Complex Transactions clauses in this ER present a resource-based approach to performing transactions that might form the basis of an extension to the OAPIF standard.

The CubeWerx server implements both simple and complex transactions and the following clause describes the implementation.

Simple transactions

Simple transactions operate on a single feature from a single collection and may be implemented using the standard HTTP PUT, POST, PATCH and DELETE methods. The specific edit endpoint for each collection can be found in the collections metadata available at:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collections

The edit endpoint is identified by the links with 'rel="edit"'. The following JSON response fragment illustrates the presence on the edit link in the collections metadata:

```
   {
     "links": [
       {
         "href":
"http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collec
tions?f=application%2Fjson",
         "rel": "self",
         "type": "application/json",
         "title": "this document"
       },
       ...
     ],
     "collections": [
       {
         "name": "DC_Building_Footprints",
         "links": [
           ...
           {
             "href":
"http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collec
tions/DC_Building_Footprints/items",
             "rel": "edit"
           },
           ...
         ],
         "extent": {
           "crs": "http://www.opengis.net/def/crs/EPSG/0/4326",
           "bbox": [
             -77.115085,
             38.810444,
             -76.909707,
             38.99561
           ]
         },
         ...
       },
       ...
     ]
   }
```


NOTE Ellipsis are used to collapse irrelevant components of the response.

In the CubeWerx server, simple transactions are supported as described in the Simple Transactions clause with the exception that the only input feature representation supported by the server for Testbed 15 is the encoding described in Geography Markup Language (GML) simple features profile, OGC 10-100r3 [http://portal.opengeospatial.org/files/?artifact_id=42729].

The following example illustrates a simple transaction that adds a new feature instance to the collection:

```
CLIENT                                                    SERVER
   |                                                        |
   |   POST /cubewerx/cubeserv/default/wfs/3.0/usbuildingfootpr|
   |        ints/collections/DC_Building_Footprints   HTTP/1.1 |
   |   Host: www.pvretano.com                               |
   |   Content-Type: application/gml+xml;version=3.2        |
   |   Accept: text/xml                                     |
   |                                                        |
   |   <?xml version="1.0" encoding="UTF-8"?>               |
   |   <DC_Building_Footprints                              |
   |      xmlns="http://www.cubewerx.com/namespaces/null"   |
   |      xmlns:gml="http://www.opengis.net/gml/3.2"        |
   |      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"|
   |      xsi:schemaLocation="http://schemas.cubewerx.com/names|
   |                         paces/null                     |
   |                         http://www.pvretano.com/cubewerx/|
   |                         cubeserv.cgi?datastore=usbuilding|
   |                         footprints&amp;service=WFS&amp;ve|
   |                         rsion=2.0&amp;request=DescribeFea|
   |                         tureType&amp;typeName=DC_Building|
   |                         _Footprints">                  |
   |      <geometry>                                        |
   |         <gml:Polygon gml:id="GID6" srsName="urn:ogc:def:cr|
   |                                          s:EPSG::4326">|
   |            <gml:exterior>                              |
   |               <gml:LinearRing>                         |
   |                  <gml:posList>38.813318 -77.025224 38.8135|
  |25 -77.025281 38.813535 -77.025397 38.813561 -77.025333 38.|
  |813637 -77.025384 38.813669 -77.025304 38.813608 -77.025197|
  | 38.813624 -77.0251 38.813775 -77.02496 38.813874 -77.02513|
  |6 38.813869 -77.025141 38.813806 -77.02515 38.813826 -77.02|
  |5184 38.81382 -77.025221 38.813765 -77.025272 38.81377 -77.|
  |025333 38.813707 -77.025488 38.813632 -77.025498 38.813526 |
  |-77.026137 38.813447 -77.026115 38.813415 -77.026145 38.813|
  |372 -77.026116 38.813369 -77.026083 38.813156 -77.026025 38|
  |.813198 -77.025772 38.813144 -77.025735 38.813127 -77.02573|
  |8 38.813109 -77.025706 38.81314 -77.025676 38.813155 -77.02|
  |564 38.813232 -77.025629 38.813295 -77.025245 38.813318 -77|
  |.025224</gml:posList>                                   |
   |               </gml:LinearRing>                        |
   |            </gml:exterior>                             |
   |         </gml:Polygon>                                 |
   |      </geometry>                                       |
   |      <name>Einstein Edifice</name>                     |
   |   </DC_Building_Footprints>                            |
   |-------------------------------------------------------->|
   |                                                        |
   |   HTTP/1.1 201 Created                                 |
   |   Location: /collections/DC_Building_Footprints/items/CWFI|
  |D.DC_BLD_FTPRINTS.0.66985                                |
```

```
      |<-----------------------------------------------------------|
```

The following examples illustrate the use of the PATCH method to update the value of a property of an existing feature. The CubeWerx server uses the encoding of the update action from WFS 2.0 (see http://docs.opengeospatial.org/is/09-025r2/09-025r2.html#283) to describe the change to be made to the feature:

```
   CLIENT                                                    SERVER
      |                                                        |
      |   POST /cubewerx/cubeserv/default/wfs/3.0/usbuildingfootpr|
      |ints/collections/DC_Building_Footprints/items/CWFID.DC_BLD_|
      |FTPRINTS.0.66985   HTTP/1.1                             |
      |   Host: www.pvretano.com                               |
      |   Content-Type: text/xml                               |
      |   Accept: text/xml                                     |
      |                                                        |
      |   <?xml version="1.0" encoding="UTF-8"?>               |
      |   <wfs:Update typeName="DC_Building_Footprints">       |
      |      xmlns:fes="http://www.opengis.net/fes/2.0"        |
      |      xmlns:wfs="http://www.opengis.net/wfs/2.0"        |
      |      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"|
      |      xsi:schemaLocation="http://www.opengis.net/wfs/2.0   |
      |      http://www.pvretano.com/schemas/wfs/2.0/wfs.xsd">    |
      |      <wfs:Property>                                    |
      |         <wfs:ValueReference>name</wfs:ValueReference>  |
      |         <wfs:Value>Madame Currie Towers</wfs:Value>    |
      |      </wfs:Property>                                   |
      |   </wfs:Update>                                        |
      |-------------------------------------------------------->|
      |                                                        |
      |   HTTP/1.1 200 OK                                      |
      |   Content-Type: application/gml+xml;version=3.2        |
      |   Location: /collections/DC_Building_Footprints/items/CWFI|
      |D.DC_BLD_FTPRINTS.0.66985                               |
      |                                                        |
      |   <?xml version="1.0" encoding="UTF-8"?>               |
      |   <DC_Building_Footprints                              |
      |      xmlns="http://www.cubewerx.com/namespaces/null"   |
      |      xmlns:gml="http://www.opengis.net/gml/3.2"        |
      |      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"|
      |      xsi:schemaLocation="http://schemas.cubewerx.com/names|
      |                          paces/null                    |
      |                          http://www.pvretano.com/cubewerx/|
      |                          cubeserv.cgi?datastore=usbuilding|
      |                          footprints&amp;service=WFS&amp;ve|
      |                          rsion=2.0&amp;request=DescribeFea|
      |                          tureType&amp;typeName=DC_Building|
      |                          _Footprints">                 |
      |      <geometry>                                        |
      |         <gml:Polygon gml:id="GID6" srsName="urn:ogc:def:cr|
```

```
|                                                 s:EPSG::4326">|
|            <gml:exterior>                                     |
|               <gml:LinearRing>                                |
|                  <gml:posList>38.813318 -77.025224 38.8135|
|25 -77.025281 38.813535 -77.025397 38.813561 -77.025333 38.|
|813637 -77.025384 38.813669 -77.025304 38.813608 -77.025197|
| 38.813624 -77.0251 38.813775 -77.02496 38.813874 -77.02513|
|6 38.813869 -77.025141 38.813806 -77.02515 38.813826 -77.02|
|5184 38.81382 -77.025221 38.813765 -77.025272 38.81377 -77.|
|025333 38.813707 -77.025488 38.813632 -77.025498 38.813526 |
|-77.026137 38.813447 -77.026115 38.813415 -77.026145 38.813|
|372 -77.026116 38.813369 -77.026083 38.813156 -77.026025 38|
|.813198 -77.025772 38.813144 -77.025735 38.813127 -77.02573|
|8 38.813109 -77.025706 38.81314 -77.025676 38.813155 -77.02|
|564 38.813232 -77.025629 38.813295 -77.025245 38.813318 -77|
|.025224</gml:posList>                                        |
|               </gml:LinearRing>                              |
|            </gml:exterior>                                    |
|         </gml:Polygon>                                        |
|      </geometry>                                              |
|      <name>Madam Currie Towers</name>                         |
|   </DC_Building_Footprints>                                   |
|<--------------------------------------------------------------|
```

Complex transactions

Complex transactions may operate on more than one feature at a time, perhaps across multiple collections, and may exhibit atomic or batch semantics. Complex transactions are implemented as described in the Complex Transactions clause with the exception that the transaction document encoding is as described in the OGC® Web Feature Service 2.0 Interface Standard, OGC 09-025r2 [http://docs.opengeospatial.org/is/09-025r2/09-025r2.html#273].

Complex transactions may be posted to the server's transaction endpoint at:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/transactions

The following diagram illustrates a complex transaction that adds a new feature to a collection, modifies an existing feature and deletes another feature from the collection:

```
CLIENT                                                  SERVER
   |                                                       |
   |   POST /transactions   HTTP/1.1                       |
   |   Host: www.pvretano.com                              |
   |   Content-Type: text/xml                              |
   |   Accept: text/xml                                    |
   |                                                       |
   |   <?xml version="1.0" encoding="UTF-8"?>              |
   |   <wfs:Transaction service="WFS" version="2.0.0"      |
   |     xmlns="http://www.cubewerx.com/cw/namespaces/null"|
   |     xmlns:cw="http://schemas.cubewerx.com/namespaces/null"|
```

```
|     xmlns:fes="http://www.opengis.net/fes/2.0"            |
|     xmlns:wfs="http://www.opengis.net/wfs/2.0"            |
|     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" |
|     xsi:schemaLocation="http://www.opengis.net/wfs/2.0    |
|                         http://www.pvretano.com/schemas/w|
|                         fs/2.0/wfs.xsd                    |
|                         http://schemas.cubewerx.com/names|
|                         paces/null                        |
|                         http://www.pvretano.com/cubewerx/|
|cubeserv.cgi?datastore=tb12&amp;service=WFS&amp;version=2.0|
|&amp;request=DescribeFeatureType&amp;typeName=DC_Building_F|
|ootprints">                                                |
|      <wfs:insert>                                         |
|         <DC_Building_Footprints>                          |
|            <geometry>                                     |
|               <gml:Polygon gml:id="GID6" srsName="urn:ogc:|
|def:crs:EPSG::4326">                                       |
|                  <gml:exterior>                           |
|                     <gml:LinearRing>                      |
|                        <gml:posList>38.813318 -77.025224 3|
|8.813525 -77.025281 38.813535 -77.025397 38.813561 -77.0253|
|33 38.813637 -77.025384 38.813669 -77.025304 38.813608 -77.|
|025197 38.813624 -77.0251 38.813775 -77.02496 38.813874 -77|
|.025136 38.813869 -77.025141 38.813806 -77.02515 38.813826 |
|-77.025184 38.81382 -77.025221 38.813765 -77.025272 38.8137|
|7 -77.025333 38.813707 -77.025488 38.813632 -77.025498 38.8|
|13526 -77.026137 38.813447 -77.026115 38.813415 -77.026145 |
|38.813372 -77.026116 38.813369 -77.026083 38.813156 -77.026|
|025 38.813198 -77.025772 38.813144 -77.025735 38.813127 -77|
|.025738 38.813109 -77.025706 38.81314 -77.025676 38.813155 |
|-77.02564 38.813232 -77.025629 38.813295 -77.025245 38.8133|
|18 -77.025224</gml:posList>                                |
|                     </gml:LinearRing>                     |
|                  </gml:exterior>                          |
|               </gml:Polygon>                              |
|            </geometry>                                    |
|            <name>Einstein Edifice</name>                  |
|         </DC_Building_Footprints>                         |
|      </wfs:insert>                                        |
|      <wfs:Update typeName="DC_Building_Footprints">       |
|         <wfs:Property>                                    |
|            <wfs:ValueReference>name</wfs:ValueReference>  |
|            <wfs:Value>Madame Currie Towers</wfs:Value>    |
|         </wfs:Property>                                   |
|         <fes:Filter>                                      |
|            <fes:PropertyIsEqualTo>                        |
|            <fes:ValueReference>name</fes:ValueReference>  |
|            <fes:Literal>Dirac Domicile</fes:Literal>      |
|            </fes:PropertyIsEqualTo>                       |
|         </fes:Filter>                                     |
|      </wfs:Update>                                        |
```

```
|      <wfs:Delete typeName="cw:DC_Building_Footprints">    |
|         <fes:Filter>                                      |
|            <fes:PropertyIsEqualTo>                        |
|            <fes:ValueReference>name</fes:ValueReference>  |
|            <fes:Literal>Madame Currie Towers</fes:Literal>|
|            </fes:PropertyIsEqualTo>                       |
|         </fes:Filter>                                     |
|      </wfs:Delete>                                        |
|   </wfs:Transaction>                                      |
|---------------------------------------------------------->|
|                                                           |
|                                                           |
|   HTTP/1.1 200 OK                                         |
|   Content-Type: application/json                          |
|                                                           |
|   <?xml version="1.0" encoding="UTF-8"?>                  |
|   <wfs:TransactionResponse xmlns:xsi="http://www.w3.org/20|
|                                    01/XMLSchema-instance"|
|      xmlns:wfs="http://www.opengis.net/wfs/2.0"           |
|      xsi:schemaLocation="http://www.opengis.net/wfs/2.0 ht|
|      tp://www.pvretano.com/schemas/wfs/2.0/wfs.xsd"       |
|      version="2.0.0">                                     |
|      <wfs:TransactionSummary>                             |
|         <wfs:totalInserted>1</wfs:totalInserted>          |
|         <wfs:totalUpdated>1</wfs:totalUpdated>            |
|         <wfs:totalReplaced>0</wfs:totalReplaced>          |
|         <wfs:totalDeleted>1</wfs:totalDeleted>            |
|      </wfs:TransactionSummary>                            |
|      <wfs:InsertResults>                                  |
|         <wfs:Feature handle="Action #1">                  |
|            <fes:ResourceId rid="CWFID.DC_BLD_FTPRINTS.0.67|
|133"/>                                                     |
|         </wfs:Feature>                                    |
|      </wfs:InsertResults>                                 |
|      <wfs:UpdateResults>                                  |
|         <wfs:Feature handle="Action #0">                  |
|            <fes:ResourceId rid="CWFID.DC_BLD_FTPRINTS.0.66|
|998"/>                                                     |
|         </wfs:Feature>                                    |
|      </wfs:UpdateResults>                                 |
|   </wfs:TransactionResponse>                              |
|<----------------------------------------------------------|
```

8.2.4. Transaction simulator

As mentioned previously, transactions must be performed on the server in order to generate activity that may then be reported using the delta update mechanism. How, specifically, transactions are performed is an important but orthogonal issue in relation to the delta updates mechanism.

In order to alleviate the client implementors in the delta updates thread from having to also build transaction capabilities, CubeWerx created a transaction simulator. The transaction simulator executes a predefined set of transactions, with random priorities and at random intervals between 0 and 3 minutes to simulate transactions activity on the TB15 CubeWerx WFS.

Because the transaction simulator will run continually for the duration of the test bed, the set of simulated transactions are, in totality, idempotent. This means that the net effect of running the transactions over time will be to add no new data to the server and thus not consume any database space.

The transaction simulator is implemented as a BASH script. The source code for the simulator is:

```
   #!/usr/bin/bash

   # Declare some variables
   declare -a priorities
   declare -a files
   declare -i sleepInterval

   # Assign the set of priorities
   priorities=(high medium low)

   # Assign the predefined set of transaction files
   files=(INS01.xml INS02.xml INS03.xml UPD01.xml DEL01.xml)

   # Loop indefinitely (or until killed!)
   while [ 1 -eq 1 ]; do

      # Run the idempotent sequence of transactions.
      for f in "${files[@]}";
      do
         echo "-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-"

         # Pick a random priority
         n=$((RANDOM % 3))
         p=${priorities[$n]}
         echo "Priority = "$p

         # POST the next Tx in the sequence to the server
         cwhttp method=POST showheaders=TRUE
url='http://www.pvretano.com/cubewerx/cubeserv?service=WFS&datastore=usbuildingfootpri
nts' extraheaders='OGC-Priority: '$p bodyfile=./$f | xmlscan indent=3 -

         # Sleep for a random interval of up to 3 minutes
         sleepInterval=$(($RANDOM % 300))
         echo "Sleeping for "$sleepInterval" seconds..."
         sleep $sleepInterval
      done
   done
```

NOTE `cwhttp` is CubeWerx's version of cUrl.

8.2.5. Examples

The clause contains example delta update requests and responses from the TB15 CubeWerx WFS. A sequence of transactions and delta update requests was used to generate these examples and the material is presented in that temporal order.

TIME 0 - Empty collection

At this time the database is empty. The following summary request:

```
http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collect
ions/DC_Building_Footprints/changesets?f=json&resultType=summary
```

generates the following response (including headers):

```
HTTP/1.1 200 OK
Date: Tue, 24 Sep 2019 20:03:39 GMT
Server: Apache
Vary: Origin
CubeWerx-Suite-Version: 9.1.19
Content-Disposition: inline; filename="Sync.json"
Cache-Control: no-cache, no-store
Pragma: no-cache
Connection: close
Content-Type: application/json
```

TIME 1 - Four mixed priority operations

At this time, 4 operations have been performed on the database; a low priority insert, a medium priority insert, and high priority insert and a low priority update.

The following summary request:

```
http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collect
ions/DC_Building_Footprints/changesets?f=json&resultType=summary
```

generates the following response (including headers):

```
HTTP/1.1 200 OK
Date: Wed, 23 Sep 2019 12:07:40 GMT
Server: Apache
Vary: Origin
CubeWerx-Suite-Version: 9.1.19
Content-Disposition: inline; filename="Sync.json"
Cache-Control: no-cache, no-store
Pragma: no-cache
Connection: close
Content-Type: application/json

{
   "summaryOfChangedItems":
   [
      {"priority":"low","count":2},
      {"priority":"medium","count":1},
      {"priority":"high","count":1}
   ]
}
```

The following delta updates request, requesting high and low priority changes:

```
http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collect
ions/DC_Building_Footprints/changesets?priority=high,low&f=json
```

generates the following response (including headers):

```
HTTP/1.1 200 OK
Date: Wed, 23 Sep 2019 12:08:18 GMT
Server: Apache
Vary: Origin
OGC-Checkpoint: cw:checkpoint:2989b850-df8d-11e9-9459-8fc4f77a6eac:DC_BLD_FTPRINTS:4
CubeWerx-Suite-Version: 9.1.19
Content-Disposition: inline; filename="Sync.json"
Cache-Control: no-cache, no-store
Pragma: no-cache
Connection: close
Content-Type: application/json

{
   "checkPoint":"cw:checkpoint:2989b850-df8d-11e9-9459-
8fc4f77a6eac:DC_BLD_FTPRINTS:4",
   "summaryOfChangedItems":
   [
      {"priority":"low","count":2},
      {"priority":"medium","count":1},
      {"priority":"high","count":1}
   ],
```

```
   "numberOfReturnedItems":3,
   "changedItems":
   [
      {"priority":"high",
       "items":
       [
          {"type":"Feature",
           "geometry":{"type":"Polygon",
                       "coordinates":
                       [
                          [
                             [-118.589772,33.027218],
                             [-118.589823,33.027295],
                             [-118.589862,33.027277],
                             [-118.589883,33.02731],
                             [-118.589986,33.027263],
                             [-118.589962,33.027226],
                             [-118.589995,33.02721],
                             [-118.58996,33.027158],
                             [-118.589896,33.027187],
                             [-118.589854,33.027124],
                             [-118.589753,33.02717],
                             [-118.589726,33.027129],
                             [-118.589641,33.027168],
                             [-118.589609,33.02712],
                             [-118.58952,33.027161],
                             [-118.589575,33.027246],
                             [-118.589614,33.027228],
                             [-118.589646,33.027276],
                             [-118.589772,33.027218]
                          ]
                       ]
           },
           "properties":{"id":"CWFID.DC_BLD_FTPRINTS.0.67057",
                         "NAME":"Einstein Edifice"}
          }
       ]
      },
      {"priority":"low",
       "items":
       [
          {"type":"Feature",
           "geometry":{"type":"Polygon",
                       "coordinates":
                       [
                          [
                             [-118.5892,33.027056],
                             [-118.589275,33.027183],
                             [-118.589439,33.027115],
                             [-118.589452,33.027136],
                             [-118.589535,33.027102],
```

```
                        [-118.589448,33.026954],
                        [-118.5892,33.027056]
                      ]
                    ]
       },
       "properties":{"id":"CWFID.DC_BLD_FTPRINTS.0.67123",
                     "NAME":"Plank Place"}
     },
     {"type":"Feature",
       "geometry":{"type":"Polygon",
                   "coordinates":
                    [
                       [
                          [-77.025331,38.81579],
                          [-77.026202,38.816265],
                          [-77.027168,38.815189],
                          [-77.027268,38.815243],
                          [-77.027395,38.815101],
                          [-77.027449,38.81513],
                          [-77.027183,38.815427],
                          [-77.027267,38.815473],
                          [-77.027069,38.815694],
                          [-77.027286,38.815812],
                          [-77.027293,38.815813],
                          [-77.027479,38.815605],
                          [-77.027644,38.815695],
                          [-77.028035,38.81526],
                          [-77.027428,38.814929],
                          [-77.027307,38.815063],
                          [-77.027291,38.814974],
                          [-77.026987,38.814809],
                          [-77.027004,38.814745],
                          [-77.027122,38.814763],
                          [-77.027168,38.814712],
                          [-77.027088,38.814668],
                          [-77.027187,38.814558],
                          [-77.026777,38.814494],
                          [-77.026751,38.814524],
                          [-77.026534,38.81449],
                          [-77.026069,38.815009],
                          [-77.02609,38.815123],
                          [-77.025804,38.815442],
                          [-77.025696,38.815383],
                          [-77.025331,38.81579]
                       ]
                    ]
       },
       "properties":{"id":"CWFID.DC_BLD_FTPRINTS.0.65956",
                     "NAME":"Heisenberg House"}
     }
   ]
```

```
      }
   ]
}
```

TIME 2 - One low priority operation

At this time, 1 low priority operation is performed on the database.

Using the checkpoint value from the previous response, the following delta update request:

```
http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collect
ions/DC_Building_Footprints/changesets/cw:checkpoint:2989b850-df8d-11e9-9459-
8fc4f77a6eac:DC_BLD_FTPRINTS:4
```

generates the following response (including headers):

```
HTTP/1.1 200 OK
Date: Wed, 23 Sep 2019 12:10:44 GMT
Server: Apache
Vary: Origin
OGC-Checkpoint: cw:checkpoint:c28d6dc8-d09b-4122-88d7-8ac6739319c4:DC_BLD_FTPRINTS:5
CubeWerx-Suite-Version: 9.1.19
Content-Disposition: inline; filename="Sync.json"
Cache-Control: no-cache, no-store
Pragma: no-cache
Connection: close
Content-Type: application/json

{"checkPoint":"OGC-Checkpoint: cw:checkpoint:c28d6dc8-d09b-4122-88d7-
8ac6739319c4:DC_BLD_FTPRINTS:5",
  "summaryOfChangedItems":
  [
      {"priority":"low","count":1}
  ],
  "numberOfReturnedItems":1,
  "changedItems":
  [
     {"priority":"low",
      "items":
      [
         {"type":"Feature",
          "geometry":{"type":"Polygon",
                      "coordinates":
                      [
                         [
                            [-78.646564,37.970954],
                            [-78.646425,37.971006],
                            [-78.646443,37.971037],
                            [-78.646352,37.97107],
                            [-78.646397,37.971146],
                            [-78.646556,37.971086],
                            [-78.64654,37.97106],
                            [-78.646612,37.971034],
                            [-78.646564,37.970954]
                         ]
                      ]
                     },
           "properties":{"id":"CWFID.DC_BLD_FTPRINTS.0.67193",
                         "NAME":"Dirac Domicile"}
         }
      ]
     }
  ]
}
```

TIME 3 - One low priority delete

At this time, a low priority delete operation is executed on the server.

Using the checkpoint value from the previous response, the following delta update request:

```
http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collect
ions/DC_Building_Footprints/changesets/cw:checkpoint:c28d6dc8-d09b-4122-88d7-
8ac6739319c4:DC_BLD_FTPRINTS:5
```

generates the following response (including headers):

```
HTTP/1.1 200 OK
Date: Wed, 23 Sep 2019 12:14:05 GMT
Server: Apache
Vary: Origin
OGC-Checkpoint: cw:checkpoint:bae9ce77-e112-45cb-8e58-526967905c97:DC_BLD_FTPRINTS:6
CubeWerx-Suite-Version: 9.1.19
Content-Disposition: inline; filename="Sync.json"
Cache-Control: no-cache, no-store
Pragma: no-cache
Connection: close
Content-Type: application/json

{"checkPoint":"cw:checkpoint:bae9ce77-e112-45cb-8e58-526967905c97:DC_BLD_FTPRINTS:6",
 "summaryOfChangedItems":
 [
    {"priority":"low","count":1}
 ],
 "numberOfReturnedItems":1,
 deletedItems":
 [
     {"priority":"low",
      "items":
      [

"http://www.pvretano.com/cubewerx/cubeserv/default/wfs/3.0/usbuildingfootprints/collec
tions/DC_Building_Footprints/items/CWFID.DC_BLD_FTPRINTS.0.65970"
      ]
     }
 ]
}
```

Chapter 9. Delta Updates Client

This section describes the implementation of the Delta Updates client. The client demonstrates a potential use case for delta updates services provided by both the Web Feature Service (WFS) and Web Processing Service (WPS).

9.1. Overview

The client is a simple application written in JavaScript, run in a Node.js v10.16.3 environment and employing the Express.js [https://expressjs.com/] framework. The client interacts with either the WFS or the WPS, depending on an argument passed at runtime.

The purpose of the client is to store a local version of a feature set as a GeoPackage that remains up-to-date with the feature set served by the WFS or WPS via changeset requests. The client ingests changesets and updates its local feature set accordingly by calling a Python script. the client also uses a PostgreSQL database to keep track of the last checkpoint for which it received a changeset.

The implementation of the component parts of the client, and rationale behind the use of different technologies, is described in detail in the sections below.

9.2. Local GeoPackage

A local feature set that mirrors the feature set hosted by the WFS/WPS is persisted in GeoPackage format (`*.gpkg`) on the same machine as the client. A GeoPackage is initialized by requesting the full feature set from the WFS/WPS. Future changes to the feature set on the WFS/WPS are described by changesets, which the client requests and parses before modifying the local GeoPackage to reflect the changes described in the changeset. In this way, the local GeoPackage can be kept up-to-date with the remote feature set.

The OGC GeoPackage format was chosen for local storage as it is a lightweight container format. The testbed participants also believed that the GeoPackage JS [https://www.npmjs.com/package/@ngageoint/geopackage] module developed by the National Geospatial-Intelligence Agency (NGA) [https://www.nga.mil] would provide a simple interface between the JavaScript server and the local GeoPackage. However, during development of the client, there were difficulties using the GeoPackage JS module within a Node.js runtime environment. This prompted the search for an alternative way for the client to modify GeoPackages.

To this end, two Python packages were chosen to interact with the local GeoPackage: `osgeo` [https://gdal.org/python/osgeo-module.html] and `Fiona` [https://fiona.readthedocs.io/en/stable/index.html]. The Express.js server spawns a Python processes in order to use these packages.

As the `osgeo` package can create a GeoPackage directly from GeoJSON, this package was used to only create the initial GeoPackage from the WFS feature set. In the absence of a GeoJSON-formatted feature set from the WPS, the initial feature set is requested as JSON and converted to GeoJSON by the client. The `osgeo` package can then read and insert the content into a GeoPackage.

`Fiona` is a package for reading and writing to a GeoPackage. Changesets received from the WFS/WPS were parsed and feature modifications, additions and deletions in the local GeoPackage were

handled by `Fiona`.

The resulting GeoPackages can be visualized in a large number of geospatial technologies.

9.3. Changeset Requests

The client requests changesets from the WFS/WPS at a set interval. Before requesting a changeset, the client queries the database to find the last checkpoint for which it has received a changeset. (Note: at this time, the client does not keep a record of which priority updates it has requested, and from which checkpoint. See Known Issues below.) If there is no checkpoint in the table that corresponds to the feature set provided by the WFS/WPS, the client will request all changes. Otherwise the client will request all changes since the checkpoint returned by the database query.

The changeset is immediately returned by the WFS endpoint whereas the WPS endpoint must be polled to determine when the process to get updates has completed. Once a changeset is received from either service, the contents are separated into updates. Updates include additions of new features, modifications to existing features, and deletions. Each changeset is written in a standardized manner into JSON files that are then passed to the Python script responsible for updating the local GeoPackage.

If an empty changeset is returned from the WFS/WPS, no action is taken by the client.

9.4. TIE Test Documentation

9.4.1. Initial Feature Set

Executed with `$ node src/server.js [wfs|wps] init`

When the `init` option is passed to the client at runtime, the client requests the initial dataset from the given service. In the case of the WFS endpoint, this feature set is received as GeoJSON and can be directly written to a local GeoPackage by the `osgeo` Python module. In the case of the WPS endpoint the data are received in plain JSON format and undergoes some slight modifications to create the GeoJSON encoding required by the Python script.

9.4.2. WFS (High Priority Updates)

Executed with: `$ node src/server.js wfs high`

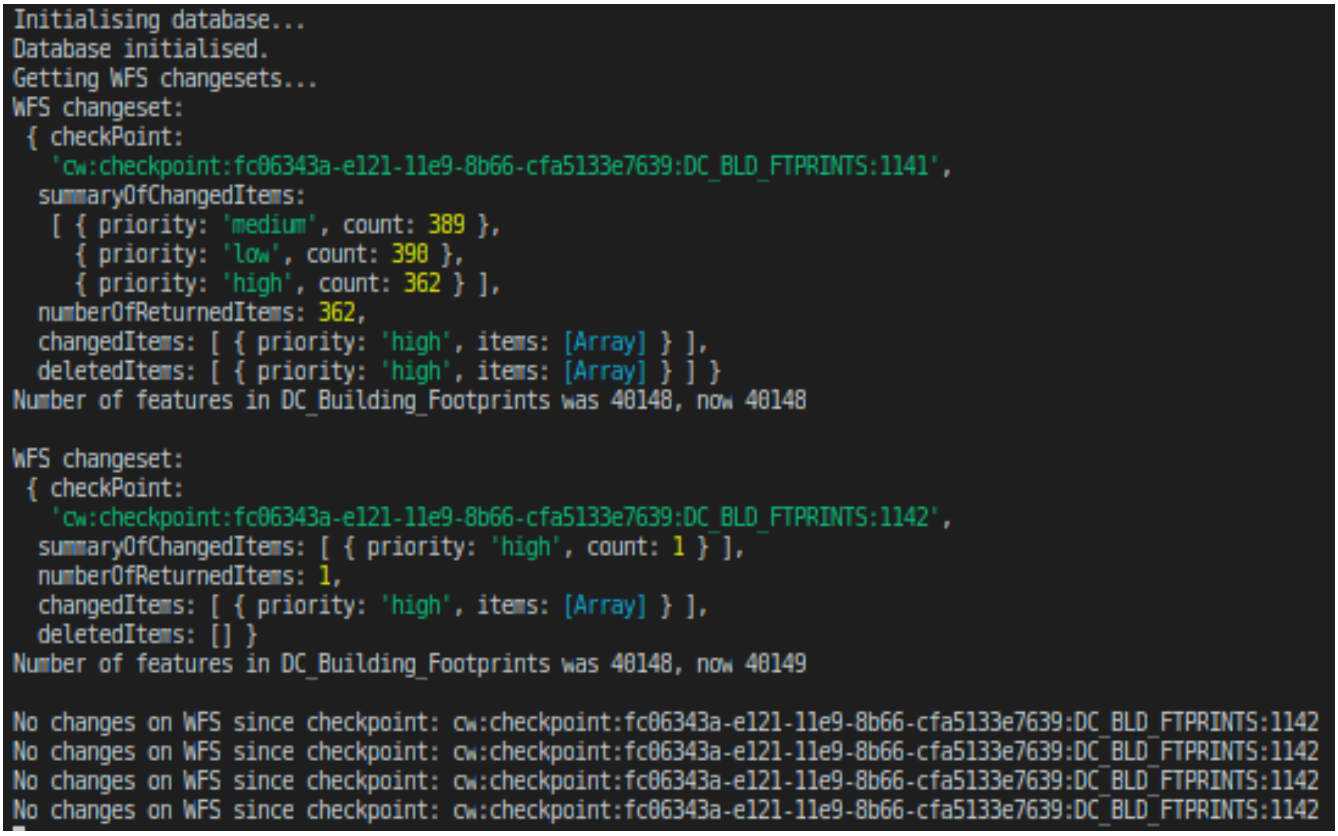


Figure 5. Client console output requesting changesets from WFS

First, the database table to keep track of the last checkpoint for which a changeset was received is initialized.

Then a request is made to the WFS endpoint for all updates since the feature set was created. In the tests performed as part of this Testbed, this results in a changeset with several hundred changes. Because the request only asked for high priority updates, only those updated features are returned in full. The client logs a summary of the changeset JSON to the console. This JSON is written to files that are passed as arguments to a Python script that makes the appropriate updates to the local feature set, logging how many features now exist compared to before the changeset was processed. The new checkpoint is retrieved from the WFS response and written to the database. The cycle then repeats after a set interval, with the new request using the new checkpoint.

Later on, an empty changeset is returned from the WFS. No action is taken, and this fact is logged to the console.

Changes to the local feature set are visualized in QGIS. The screenshot below shows the entire DC_Building_Footprints feature set with the tail end of the attribute table displayed:

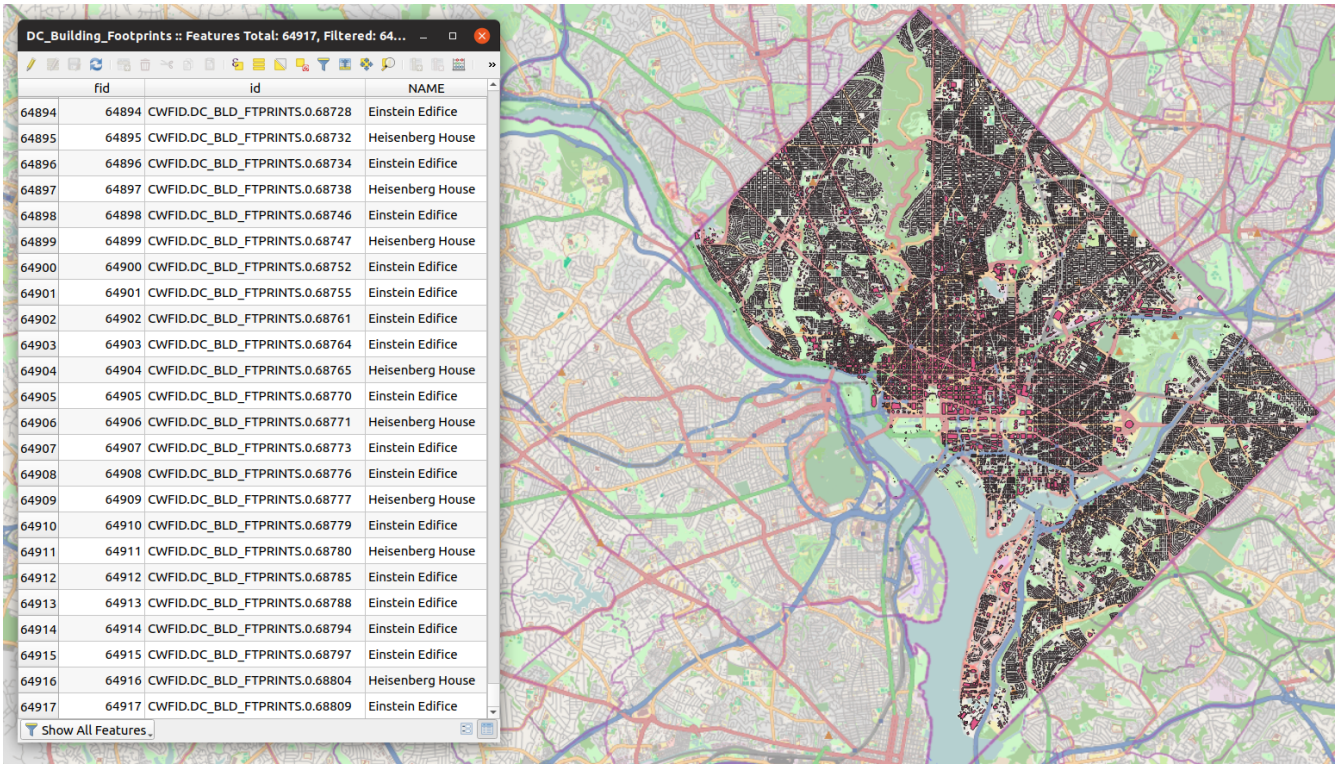


Figure 6. The entire DC_Building_Footprints feature set with the tail end of the attribute table displayed

After a changeset has been received, the user sees the same feature set with the additional feature, Madame Currie Towers, shown in the attribute table:

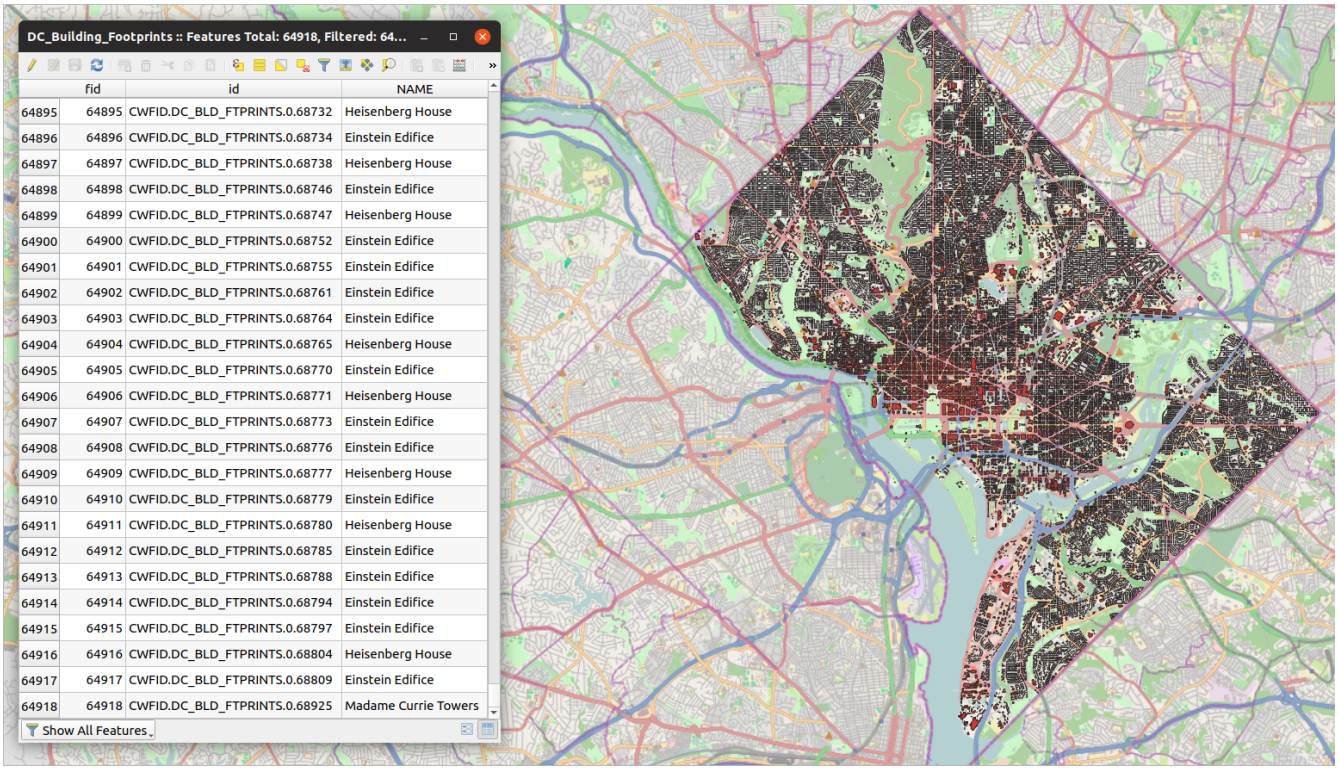


Figure 7. Feature set with the additional feature presented to the user Madame Currie Towers

A little while later, more updates are received:

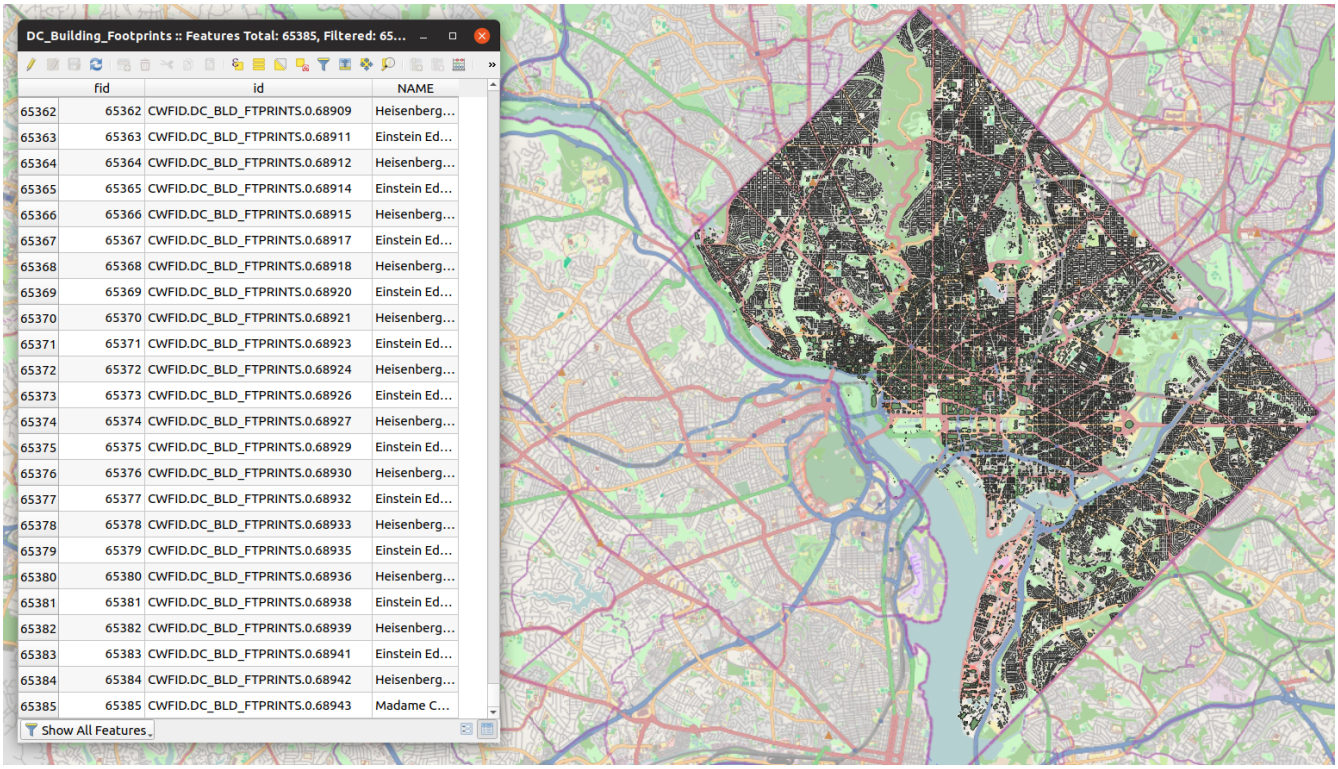


Figure 8. Additional updates received

This update (changeset) contains a deletion. Note that the final entry in the attribute table (FID 65385) has been removed and the total number of features (found at the top of the attribute table window) was reduced by one.

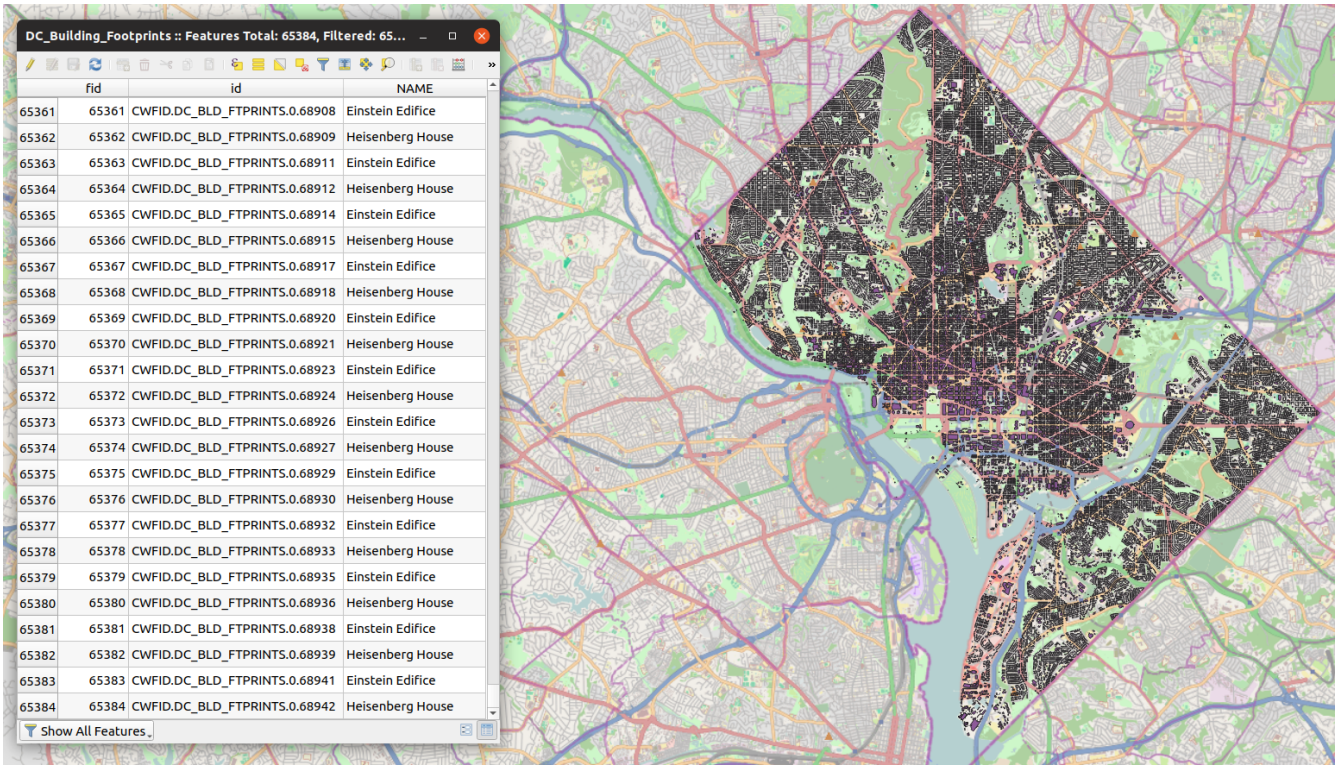


Figure 9. An update containing a deletion

9.4.3. WPS (High Priority Updates)

Executed with: `$ node src/server.js wps high`

```
Initialising database...
Database initialised.
Getting changeset from WPS...
No changes on WPS since beginning.
WPS changeset:
 { inlineValue:
   { checkPoint: 'f9e698ba-491a-4d8b-8b8b-21fb691af147',
     summaryOfChangedItems: [ [Object], [Object], [Object] ],
     numberOfReturnedItems: 1,
     changedItems: [ [Object] ] } }
Number of features in tbl5-du:DistrictofColumbia was 58326, now 58327

child process exited with code 0
No changes on WPS since checkpoint: f9e698ba-491a-4d8b-8b8b-21fb691af147.
No changes on WPS since checkpoint: f9e698ba-491a-4d8b-8b8b-21fb691af147.
No changes on WPS since checkpoint: f9e698ba-491a-4d8b-8b8b-21fb691af147.
No changes on WPS since checkpoint: f9e698ba-491a-4d8b-8b8b-21fb691af147.
WPS changeset:
 { inlineValue:
   { checkPoint: '06b400ac-db0f-4975-8cd2-b5838ccd1bff',
     summaryOfChangedItems: [ [Object], [Object], [Object] ],
     numberOfReturnedItems: 1,
     changedItems: [ [Object] ] } }
Number of features in tbl5-du:DistrictofColumbia was 58327, now 58328

child process exited with code 0
No changes on WPS since checkpoint: 06b400ac-db0f-4975-8cd2-b5838ccd1bff.
```

Figure 10. Client console output requesting changesets from WPS

The client has similar output when interacting with the WPS endpoint as it does for the WFS endpoint. First, the database is initialized, then changes are requested from the WPS.

Initially, there are no changes. The client repeats its request for changes at regular intervals, sometimes receiving a populated changeset and other times receiving an empty changeset. A populated changeset is parsed and changes to features are passed to the same Python script used by the WFS instance in order to update the local GeoPackage. The new checkpoint is stored in the database, ready to be used for the next changeset request.

The initial feature set as received from the WPS was displayed using QGIS:

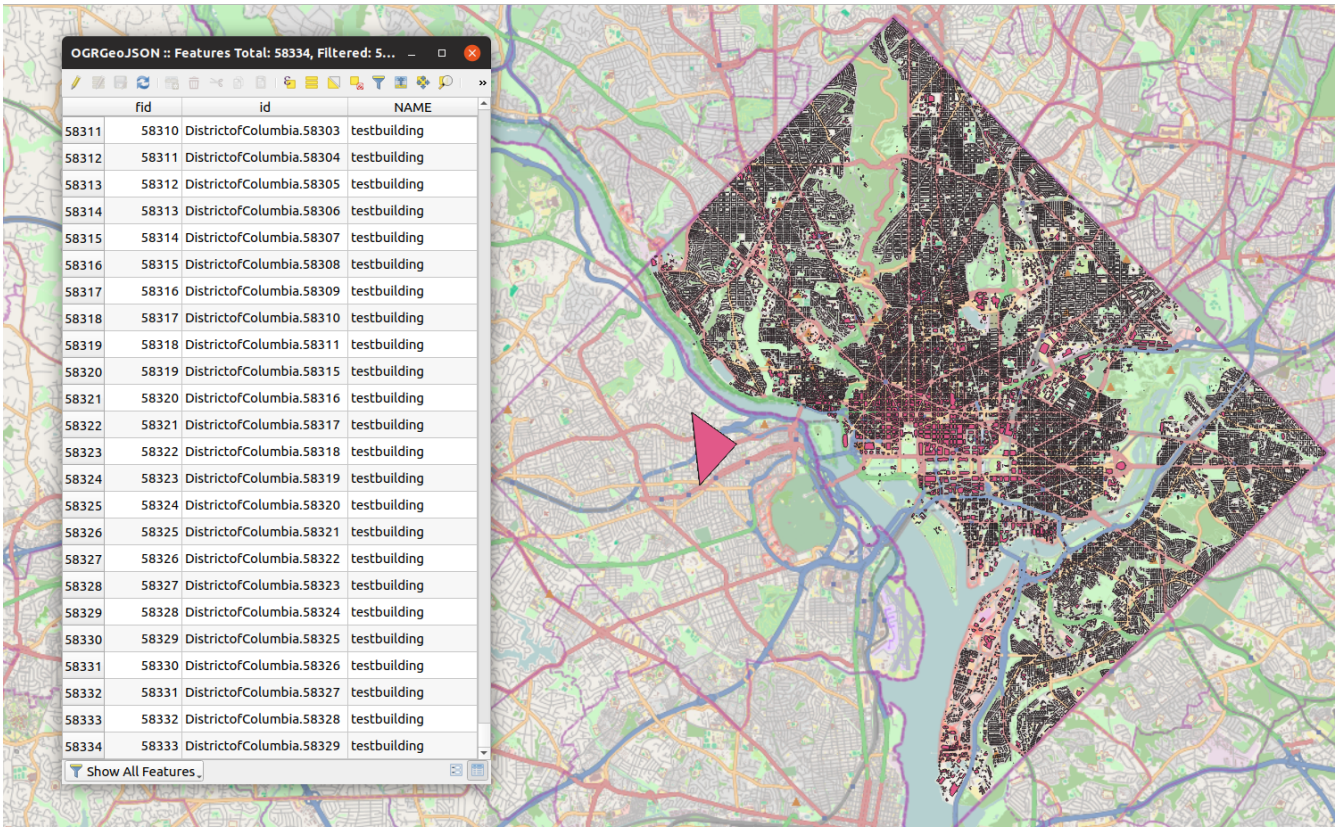


Figure 11. The initial feature set received from the WPS and displayed using QGIS

Additions to the feature are noticeable in the attribute table:

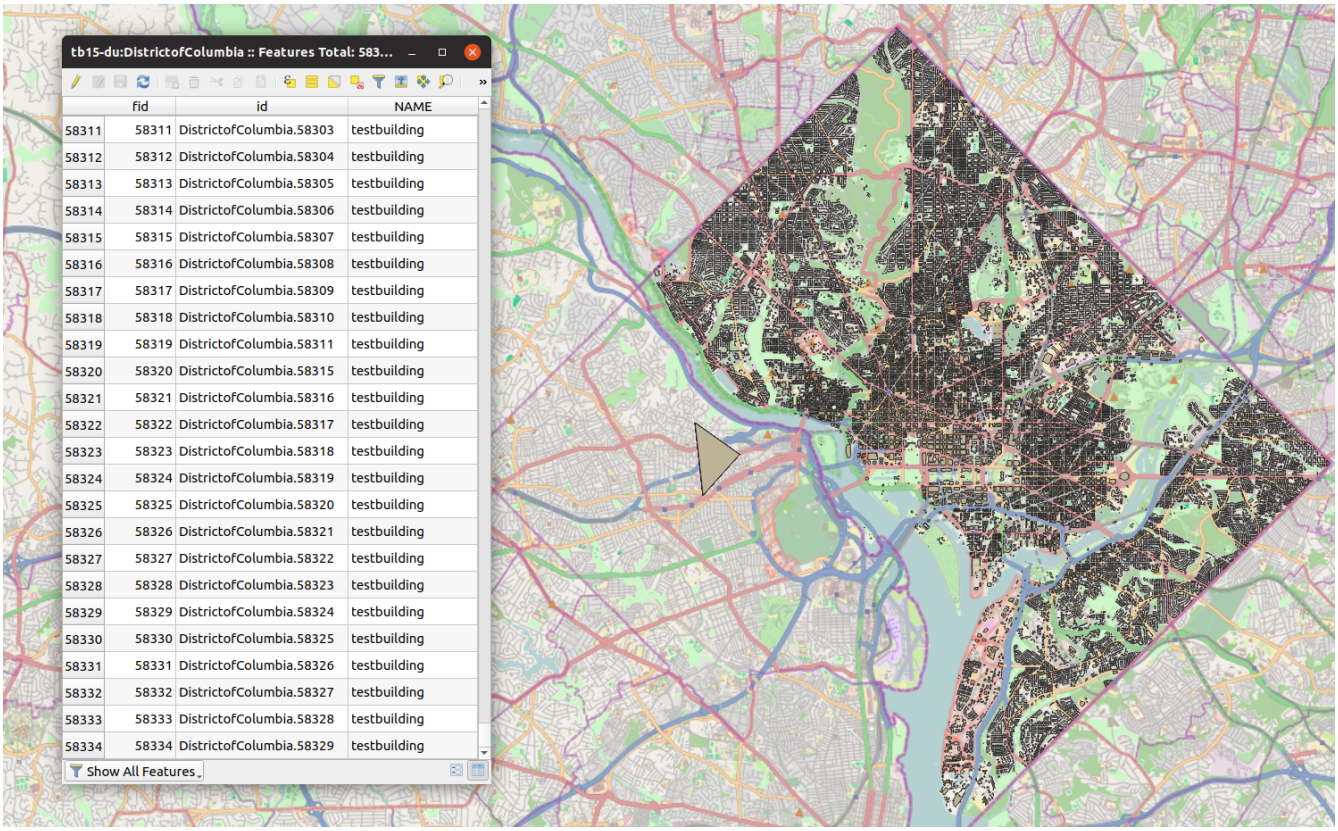


Figure 12. Additions to the feature are noticeable in the attribute table

The same feature later being modified is also evident:

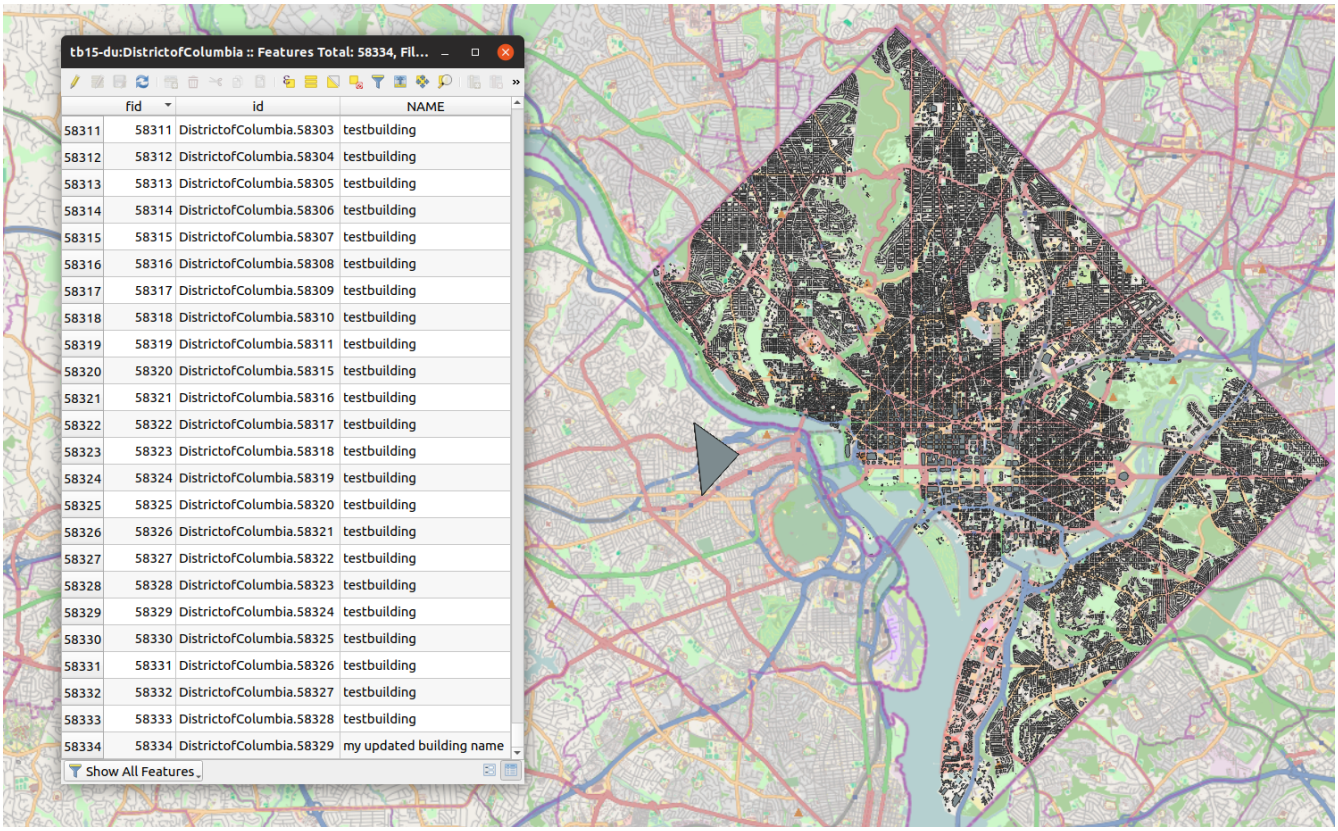


Figure 13. The feature being modified

Another feature is deleted as shown in the console:

```
No changes on WPS since checkpoint: fd9f3108-a484-4567-95f4-4c9b5215c80a.
No changes on WPS since checkpoint: fd9f3108-a484-4567-95f4-4c9b5215c80a.
WPS changeset:
 { inlineValue:
   { checkPoint: 'af69bac6-6893-4e81-a3aa-7d5a68c822d4',
     summaryOfChangedItems: [ [Object], [Object], [Object] ],
     numberOfReturnedItems: 1,
     deletedItems: [ [Object] ] } }
Number of features in tb15-du:DistrictofColumbia was 58334, now 58333

child process exited with code 0
No changes on WPS since checkpoint: af69bac6-6893-4e81-a3aa-7d5a68c822d4.
```

Figure 14. Deleted feature shown in the console

And in the attribute table in QGIS:

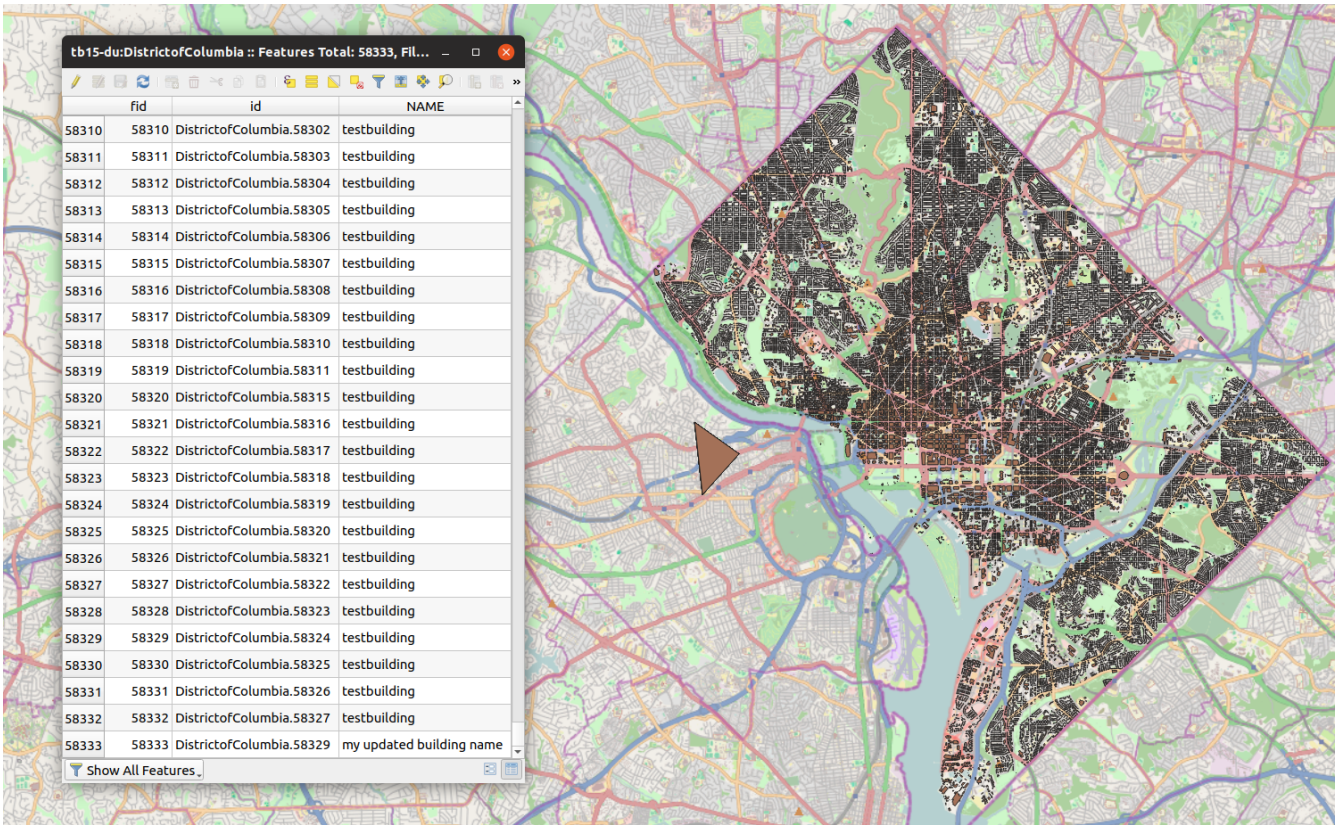


Figure 15. Deleted feature shown in QGIS

The same procedure for both WFS and WPS can be run with any single or combination of updates priorities, e.g. with `$ node src/server.js wps high,medium`.

9.5. Known Issues

The client, particularly in its interactions with the WPS endpoint, is susceptible to still writing modifications from one changeset to the local GeoPackage when another changeset is received. This can lead to `Fiona` trying to write to a GeoPackage that is open elsewhere, which will throw an error. Using a 30 second interval between requests to either service reduces the likelihood of this occurring is advised.

9.6. Recommendations

Currently, the client is partially stateful. The client keeps a record of the last checkpoint for which it has requested a changeset and so is able to request only changes since that point. This reduces the amount of redundant data transferred between the client and the WFS/WPS. However, due to time limitations, it was not possible to make the client fully stateful. For example, the client does not record which priority updates it requested from which checkpoint. A future iteration of the client would need to persist this information so that it can request low priority updates from the last checkpoint from which it received low priority updates, etc.

Additionally, this client does not have the ability to change the priority of updates requested partway through running. The priority of updates to request is set on the command line at runtime. A further avenue for development could be to enable the client to make decisions about which priority updates to request based on its own environment and connection to the WFS/WPS. A simulated poor connection to the WFS/WPS, for example, might result in the client requesting only

high priority updates until the connection improves.

Chapter 10. Conclusion

This ER concludes that prioritized delta updates of datasets can be served using a transactional extension of the OGC API – Features as well as a WPS/OGC API – Processes implementation in front of WFS instances.

The approach using WPS as facade in front of WFS instances works well. This is especially true when implementing the OGC API – Processes as the requests and responses are very similar to the ones specified by the extension to the OGC API – Features. Further, the same algorithm was used to keep track of the changes. The approach worked with an unmodified GeoServer WFS-T. There are however some shortcomings when using a WPS facade:

- General overhead of a proxy (parsing of requests/generating of responses, communication with the downstream WFS-T).
- If the OGC API – Processes is utilized, including requests/responses encoded in JSON, they have to be transformed to XML, as the WFS-T uses XML for requests and responses.
- No simple transactions are possible.

10.1. Topics for future work

10.1.1. The handling of delta updates in simulated DDIL environments

The client used for the TIE tests was running in a desktop environment with stable internet connection. To investigate the usefulness of the delta updates mechanisms in DDIL environments, unstable internet connections could be simulated either using software or hardware mechanisms.

10.1.2. Context-based prioritization, for example using a mobile client that only needs high priority updates

For the experiments in this ER, the prioritization of the delta updates was specified as parameter of the request. Investigation as to how a client can indicate limited bandwidth ahead of the request so only high priority updates are delivered is recommended.

10.1.3. Investigate a common base for delta updates in OGC APIs

The retrieval of delta updates/changesets is a topic in several other ERs, for example in the OGC Testbed-15: Open Portrayal Framework Engineering Report (OGC 19-018) [2] and OGC Testbed-15: Images and ChangesSet API Draft Specification (OGC 19-070) [3]. Investigation of what are the similarities and differences and whether a common approach can be identified is recommended.

Appendix A: JSON Schema Listings

NOTE	These listings are in an OpenAPI 3.0 flavored JSON schema.

changeset.json

```
{
    "description": "a document containing the delta updates since a specified
checkpoint",
    "type": "object",
    "required": [
        "summaryOfChangedItems"
    ],
    "properties": {
        "checkPoint": {
            "description": "the checkpoint value",
            "type": "string"
        },
        "summaryOfChangedItems": {
            "description": "a per-priority list of change counts",
            "type": "array",
            "items": {
                "description": "a count of changes corresponding to a specific
priority label",
                "type": "object",
                "required": [
                    "priority",
                    "count"
                ],
                "properties": {
                    "priority": {
                        "description": "the priority label",
                        "type": "string"
                    },
                    "count": {
                        "description": "the count of changes tagged with the
corresponding priority label",
                        "type": "integer"
                    }
                }
            }
        },
        "numberOfReturnedItems": {
            "description": "the number of changed items that are presented in this
response document; this may be less than the total number of changes",
            "type": "integer"
        },
        "changedItems": {
            "description": "a list of new or modified resources",
            "type": "array",
```

```
            "items": {
                "description": "a representations of or reference to the changed
resource; e.g. a GeoJSON-encoded feature",
                "type": "object",
                "required": [
                    "items"
                ],
                "properties": {
                    "priority": {
                        "description": "a priority label",
                        "type": "string"
                    },
                    "items": {
                        "type": "array",
                        "items": {
                            "oneOf": [
                                {
                                    "type": "object"
                                },
                                {
                                    "$ref": "link.json"
                                }
                            ]
                        }
                    }
                }
            }
        },
        "deletedItems": {
            "description": "a list of identifiers of deleted resources",
            "type": "array",
            "items": {
                "description": "the identifier of a deleted feature",
                "type": "object",
                "required": [
                    "items"
                ],
                "properties": {
                    "priority": {
                        "description": "a priority label",
                        "type": "string"
                    },
                    "items": {
                        "type": "array",
                        "items": {
                            "type": "string"
                        }
                    }
                }
            }
        }
```

```
    }
}
```

link.json

```
{
    "type": "object",
    "required": [
        "href"
    ],
    "properties": {
        "href": {
            "type": "string"
        },
        "rel": {
            "type": "string",
            "example": "service"
        },
        "type": {
            "type": "string",
            "example": "application/json"
        },
        "hreflang": {
            "type": "string",
            "example": "en"
        },
        "title": {
            "type": "string"
        }
    }
}
```

patch.json

```
{
    "type": "object",
    "properties": {
        "add": {
            "type": "array",
            "items": {
                "$ref": "nameValuePair.json"
            }
        },
        "modify": {
            "type": "array",
            "items": {
                "$ref": "nameValuePair.json"
            }
        },
        "remove": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    }
}
```

transaction.json

```
{
    "type": "object",
    "required": [
        "transaction"
    ],
    "properties": {
        "semantic": {
            "type": "string",
            "enum": [
                "atomic",
                "batch"
            ],
            "default": "atomic"
        },
        "transaction": {
            "$ref": "transaction-prop.json"
        }
    }
}
```

transaction-prop.json

```
{
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "oneOf": [
                {
                    "$ref": "insert-action.json"
                },
                {
                    "$ref": "replace-action.json"
                },
                {
                    "$ref": "update-action.json"
                },
                {
                    "$ref": "delete-action.json"
                }
            ]
        }
    }
}
```

Appendix B: Revision History

Table 13. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
2019-04-26	B. Pross	.1	all	initial version
2019-05-21	B. Pross	.1	1,5	Work on Summary, Add section 5 WPS implementation
2019-07-31	B. Pross	.1	5	Add images
2019-08-15	B. Pross	.1	all	Transfer content from wiki
2019-09-11	C. Reed	.1	All	Initial internal review.
2019-10-01	E. Ousby	.1	9	Add client documentation.
2019-10-23	B. Pross	.1	All	Incorporate comments from review.
2019-12-06	C. Reed	.1	1,2,6,8,9,10	Final internal review.
2019-12-13	G. Hobona	.1	all	Final OGC staff review.

Appendix C: Bibliography

1. Pross, B.: OGC API – Processes (formerly named OGC WPS 2.0 REST/JSON Binding Extension), https://rawcdn.githack.com/opengeospatial/wps-rest-binding/master/docs/18-062.html.
2. Klopfer, M.: OGC Testbed-15: Open Portrayal Framework Engineering Report. OGC 19-018,Open Geospatial Consortium, http://docs.opengeospatial.org/per/19-018.html .
3. Pau, J.M.: OGC Testbed-15:Images and ChangesSet API Engineering Report. OGC 19-070,Open Geospatial Consortium, http://docs.opengeospatial.org/per/19-070.html .