

OGC Testbed-16
DGGS and DGGS API Engineering Report

Publication Date: 2021-01-13

Approval Date: 2020-12-14

Submission Date: 2020-11-19

Reference number of this document: OGC 20-039r2

Reference URL for this document: <http://www.opengis.net/doc/PER/t16-D017>

Category: OGC Public Engineering Report

Editor: Robert Gibb, Byron Cochrane, Matthew Purs

Title: OGC Testbed-16: DGGS and DGGS API Engineering Report

OGC Public Engineering Report

COPYRIGHT

Copyright © 2021 Open Geospatial Consortium. To obtain additional rights of use, visit <http://www.opengeospatial.org/>

WARNING

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

LICENSE AGREEMENT

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

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

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

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

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

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

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the

Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Table of Contents

1. Subject	7
2. Executive Summary	8
2.1. Background and Expectations of the Testbed-16 DGGs thread	8
2.2. Summary of work undertaken	8
2.3. Highlights from the Testbed-16 participants	9
2.4. Overview of recommendations	10
2.5. Document contributor contact points	11
2.6. Foreword	11
3. References	12
4. Terms and definitions	13
4.1. Abbreviated terms	17
5. Overview	18
6. DGGs and DGGs Reference System Selection	19
6.1. Semantics for DGGs libraries and their association with the current DGGs draft standards	19
6.1.1. DGGs RS provider	19
6.1.2. DGGs RS navigator	19
6.2. The question of DGGs data	20
6.3. DGGs libraries	20
6.4. DGGs Libraries selected by the Testbed-16 DGGs thread	23
6.5. DGGs Reference systems selected by the Testbed-16 DGGs thread	23
7. DGGs API	26
7.1. Options for DGGs and OGC API	26
7.2. Aligning DGGs to OGC API - Features	26
7.3. Aligning DGGs to OGC API - Process	26
7.4. Implementation in the deliverables	27
7.5. Treatment of geometry	27
7.6. Operations specified for ZoneQuery	28
7.7. OGC API - Features for DGGs description	30
7.7.1. API Features implementation	31
7.7.2. API Features instances	39
7.7.3. API functionality	39
7.7.4. Supporting Assets	41
7.8. OGC API - Processes for DGGs description	41
7.8.1. Basic ideas:	41
7.8.2. Formats:	42
7.8.3. Questions, discussion topics:	42
7.8.4. Endpoints	42
7.8.5. Models	56

7.9. Is there a need for an OGC API DGGs implementation standard?	64
7.9.1. Proposal for OGC WKT for DGGs geometries	65
8. Use Cases	70
8.1. Scope of Works under the Testbed-16 CFP	70
8.1.1. Use Case #1 - GPS Location to DGGs Cell	70
8.1.2. Use Case #2 - COVID-19 Active Cases Near Me	70
8.1.3. Use Case #3 - Bushfire Impacts from the "Black Summer" Bushfires in Australia	71
8.1.4. Use Case #4 - A DGGs version of a DAPA Use Case making use of one or more Jupyter notebooks	71
9. DGGs Server and API	73
9.1. DGGs library comparisons and choices	73
9.2. DGGs Java API	78
9.3. The DGGs geometry store	78
9.4. The ClickHouse storage choice	83
9.5. Importing data in ClickHouse	87
9.6. The ClickHouse DGGs data store	89
9.7. Displaying false color maps of DGGs data	93
9.8. GeoServer DGGs API	96
9.9. GeoServer DGGs based DAPA API	107
9.9.1. The API, HTML representations, and process resources	108
9.9.2. Notes on implementation and performance	114
10. DGGs Demo Client	118
10.1. Background - Choice to create a native DGGs viewer	118
10.2. Theory and logic behind the native DGGs viewers	118
10.2.1. Examples	119
10.3. Client implementation	120
10.4. Viewers	120
10.4.1. How do the viewers work? Paint by numbers	121
10.4.2. PyDGGin.py	121
10.5. Jupyter Notebook DGGs	124
10.6. Data	124
10.7. DGGs API Queries	128
10.8. Client Enhancements and Future Work	129
11. DGGs Enabled Data Services	131
11.1. OGC API - Features instances	131
11.1.1. TB16Pix Ref	131
11.1.2. Geofabric	132
11.1.3. ASGS	133
11.2. OGC API - Features architecture	134
11.3. Supporting Assets	135
11.3.1. Supporting software tools	135

11.3.2. Semantic Assets	136
12. Future Tasks	141
12.1. Maturing DGGs Reference libraries to meet community & future testbed needs	141
12.1.1. Development tasks identified for H3	141
12.1.2. Development tasks identified for rHEALPix	142
12.1.3. Development tasks identified for both H3 and rHEALPix	142
12.1.4. Development tasks relevant to other DGGs libraries	142
12.2. OGC API(s) for DGGs	143
12.3. DGGs processing opportunities	143
12.3.1. Pre-built multi-resolution statistics	143
12.3.2. Just in time precision	144
12.4. DGGs Analytics - What Does that Really Mean?	145
12.5. The evolution of the OGC DGGs Registry	145
12.6. Opportunities for DGGs API in Interoperability Experiments	146
Appendix A: Revision History	148
Appendix B: Bibliography	149

Chapter 1. Subject

This OGC Testbed-16 Engineering Report (ER) documents the needs and key requirements for drafting an OGC Discrete Global Grid Systems (DGGS) Application Programming Interface (API) standard. The draft DGGS API is defined using the OpenAPI 3.0 specification. The work documented in this ER represents the beginning of a multi-initiative process to fully realize the benefits of standards compliant DGGS implementations and to help drive adoption of DGGS as a key element in advanced Spatial Data Architectures. The Testbed participants investigated a Client-Server DGGS architecture involving one (or more) DGGS Server implementations, DGGS-enabled Data Sources and a simple front-end DGGS Client. DGGS API functionality will be tested using one (or more) simple use case scenarios focusing on the two-way translation between geographic locations and DGGS Zonal Identifiers.

[1] <https://modwsgi.readthedocs.io>

[2] <https://pypi.org/project/Werkzeug/>

[3] <https://pypi.org/project/Flask/>

[4] <https://pypi.org/project/pyldapi/>

[5] <https://w3id.org/dggs/ogcldapi>

[6] <https://pypi.org/project/rdfliib/>

[7] <https://pypi.org/project/rHEALPixDGGS/>

[8] <https://graphdb.ontotext.com>

[9] <https://github.com/surroundaustralia/dggsgv>

Chapter 2. Executive Summary

2.1. Background and Expectations of the Testbed-16 DGGs thread

A Discrete Global Grid System (DGGs) represents a spherical partitioning of the Earth's surface into a grid of cells (or zones) (Wikipedia). The OGC Members approved and maintain an Abstract Specification (AS) that captures the foundational concepts for DGGs ([OGC 15-104r5](https://docs.opengeospatial.org/as/15-104r5/15-104r5.html)) [<http://docs.opengeospatial.org/as/15-104r5/15-104r5.html>]. This Testbed task aims to begin the process to move towards an OGC Implementation Standard for DGGs through the creation of open-source DGGs reference implementations. Testbed-16 represents the initial effort of what is considered a multi-initiatives process.

DGGs offer a new way for geospatial information to be stored, visualized, and analyzed. Based on a partitioning of the Earth's surface into a spherical grid, DGGs allows geospatial information to be represented in a way that more intuitively reflects relationships between data and the Earth's surface. With DGGs, providers and consumers of geospatial information can eliminate many of the uncertainties and distortions inherently present with traditional coordinate systems. To fully realize the benefits of DGGs, standard-compliant implementations are required to allow zone-ID management across DGGs with varying structure and alignment.

DGGs presents an opportunity for the geospatial community to implement a representation of Earth that is vastly different from traditional coordinate system based approaches. DGGs has the potential to enable storage, analysis and visualization of geospatial information in a way that more accurately reflects the relationship between data and the Earth. While the OGC DGGs Abstract Specification captures fundamental DGGs concepts, the Testbed-16 DGGs thread initiated work to more concretely demonstrate DGGs in order to drive adoption. This includes advancement towards development of a DGGs reference implementation.

Key questions addressed by the work include:

- What DGGs structure would be best for developing a reference implementation? For example, Uber's Hexagonal Hierarchical Spatial Index or the Open Equal Area Global Grid (OpenEAGGR)
- What is a simple application that could be used to demonstrate the value of the reference implementation?
- What should be considered for future work oriented towards operational implementation of DGGs?

2.2. Summary of work undertaken

The participants in the Testbed-16 DGGs thread approached these questions by:

1. Undertaking a review of what is needed from a DGGs library ([DGGs and DGGs Reference System Selection](#)) and introducing new terminology ([Terms and definitions](#)) to distinguish the different roles DGGs libraries can perform.
2. Undertaking a review of existing open-source DGGs libraries against these roles and selecting

two for use in the Testbed ([DGGS and DGGS Reference System Selection](#)).

- a. Uber's H3 library
 - b. Manaaki Whenua's rHEALPix library
3. Identifying candidate Use Cases, ideally that were both aligned with other Testbed-16 threads, and achievable with the current DGGS libraries and resources available to participants. ([Use Cases](#))
 4. Identifying which OGC API would best demonstrate DGGS ([DGGS API](#))
 5. Defining DGGS variants of OGC API - Features ([OGC API - Features for DGGS description](#)) and OGC API - Processes ([OGC API - Processes for DGGS description](#)).
 6. Developing server implementations and standing up server instances of each API:
 - a. OGC API - Features ([DGGS Enabled Data Services](#)) as a native DGGS server,
 - b. OGC API - Records ([DGGS Enabled Data Services](#)) as a discovery mechanism for the feature services, and
 - c. OGC API - Processes ([DGGS Server and API](#)) on a native DGGS datastore wrapped by GeoServer.
 7. Populating each server with data covering the Australian Capital Territory (ACT) appropriate for the final Use Case.
 - a. DGGS native data for Australian Statistical Area polygons and River Catchment polygons served through OGC API - Features ([DGGS Enabled Data Services](#)), and
 - b. DGGS native Sentinel 2 data and processes to generate NDVI, NDBI & NDWI and band statistics served through OGC API - Processes ([DGGS Server and API](#)).
 8. Developing native DGGS desktop and Jupyter Notebook demonstration clients to interact with the two API ([DGGS Demo Client](#)).
 9. Exercising Jupyter Notebook client against the servers to show the Use Case in action.
 10. Mapping the Testbed-16 Data Access and Processing API (DAPA) onto the DGGS OGC API - Processes, to demonstrate a DGGS variant of the Testbed-16 DAPA ([GeoServer DGGS based DAPA API](#)).
 11. Reviewing participants experiences to assemble and prioritise recommendations for future work ([Future Tasks](#)).

2.3. Highlights from the Testbed-16 participants

Although DGGS implementations have been deployed in previous testbeds, this was the first time that a DGGS thread has appeared in an OGC Testbed. As a consequence, there were widely differing expectations both across the participants and between the participants and the sponsors. However, once all the participants had gained a common understanding of DGGS, progress was remarkably fast and a lot simpler than was anticipated.

Key highlights from this effort include:

- Implementation of DGGS as a collection of virtual vector layers, allowing users to easily access various DGGS resolutions through formats best suited to their needs.

- Demonstration of the ability of DGGs to incorporate widely used OGC standard services, allowing organizations to leverage past data and infrastructure investments in a new way.
- Fast development of a robust Earth observation-oriented application, demonstrating the ability of DGGs to quickly and simply enable forms of analysis that are highly complex undertakings with traditional geospatial analysis techniques.

Both server implementations adopted a common strategy of wrapping the chosen DGGs libraries into their systems to present the DGGs Reference System (RS) as a collection of virtual vector layers. Each layer (or item in the collection) corresponds to a level of the DGGs RS grid hierarchy. In both API implementations this allowed the client to select whether it wanted a GeoJSON or a JSON payload. With the GeoJSON payload, the DGGs library created the necessary coordinates for each zone on-the-fly. With the JSON payload, a native DGGs geometry was supplied comprised only of DGGs zone-IDs (the unique identifier associated with each zone).

For the GeoServer OGC API - Processes implementation this means that all the native GeoServer vector processing functionality, and all the existing GeoServer services - such as OGC WMS and OGC WFS - are exposed to traditional GeoServer clients to use DGGs data. A single GeoServer DGGs wrapper library was created to deliver both H3 and rHEALPix solutions. The no-SQL database ClickHouse was chosen as the GeoServer backend datastore for the DGGs data. This proved very fast for delivering Sentinel 2 data and generating both statistical summaries and index calculations (e.g. NDVI, NDBI & NDWI) on Sentinel 2 data. In the backend ClickHouse datastore, the DGGs zone ID was the primary index for the pixels of Sentinel 2 data.

In the OGC API - Features implementation, a linked data approach was taken using an Resource Description Framework (RDF) datastore. This proved very successful and this Testbed experience is already feeding into the OGC GeoSPARQL Standard roadmap. Vector data for Australia's Level 1 Statistical Meshblocks and Hydrological Catchments were converted to DGGs RDF data with the DGGs Cell-ID as the predicate (i.e primary index) for the data. The OGC API - Features end-point presented this data as either JSON Linked Data (JSON LD) - using the DGGs zone -ID as the only representation of geometry, or as GeoJSON with the DGGs library generating vector geometries on-the-fly as required.

Participants started with an expectation that the Testbed-16 thread would deliver 'a simple application'. That participants were able to stand up an application delivering native DGGs vector and earth observation capability to users of both traditional Geographic Information Systems (GIS) clients and to native DGGs clients reinforced the versatility of DGGs and underscored its promise to deliver rapid multi-disciplinary analyses. Systems based on DGGs have been shown to have the characteristics of a chameleon that can acquire the character of both vector GIS systems and of earth observation systems. This means that both raster and vector data can be served through either raster or vector or DGGs native services. In its native form, DGGs also offer additional analytical versatility that is not present in either of these.

2.4. Overview of recommendations

Testbed participants documented future tasks in the following areas:

1. Maturing DGGs Reference Library Implementations to bring them into conformance with OGC Topic 21 v2.0 ([Maturing DGGs Reference libraries to meet community & future testbed needs](#));

2. Drafting and elaboration of OGC APIs for DGGs ([OGC API\(s\) for DGGs](#));
3. Exploring the opportunities and limitations of DGGs driven analytics ([DGGs processing opportunities](#));
4. Implementation of OGC Registries for DGGs Implementations and DGGs enabled data services ([The evolution of the OGC DGGs Registry](#));
5. Targeted DGGs Interoperability Experiments ([Opportunities for DGGs API in Interoperability Experiments](#)).

2.5. Document contributor contact points

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

Contacts

Name	Organization	Role
Byron Cochrane	OpenWork Ltd	Editor
Robert Gibb	Manaaki Whenua Landcare research	Editor
Matthew Purss	Pangaea Innovations Pty. Ltd.	Editor
Adrian Cochrane	OpenWork Ltd	Contributor
Nicholas J. Car	SURROUND Australia Pty Ltd	Contributor
Andrea Aime	GeoSolutions S.A.S.	Contributor

2.6. 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 06-121r9, OGC® Web Services Common Standard](https://portal.opengeospatial.org/files/?artifact_id=38867&version=2) [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC: [Topic 21: Discrete Global Grid Systems Abstract Specification](http://docs.opengeospatial.org/as/15-104r5/15-104r5.html) [http://docs.opengeospatial.org/as/15-104r5/15-104r5.html]
- OGC: [Draft Topic 21 v2 - Discrete Global Grid Systems - Part 1 Core Reference system and Operations and Equal Area Earth Reference System abstract Specification](https://portal.ogc.org/files/?artifact_id=93412&version=1) [https://portal.ogc.org/files/?artifact_id=93412&version=1]
OGC Topic 21 v2 is identical in normative content to N5348 ISO/DIS 19170-1, whose DIS Ballot was approved on 2020-Oct-10 with only minor editorial comments.
- OGC: [Draft OGC API - Common - Part 1: Core](https://htmlpreview.github.io/?https://github.com/opengeospatial/oapi_common/blob/master/19-072.html) [https://htmlpreview.github.io/?https://github.com/opengeospatial/oapi_common/blob/master/19-072.html]
- OGC: [Draft OGC API - Common - Part 1: Collections | Geospatial Data](https://htmlpreview.github.io/?https://github.com/opengeospatial/oapi_common/blob/master/20-024.html) [https://htmlpreview.github.io/?https://github.com/opengeospatial/oapi_common/blob/master/20-024.html]
- Open Geospatial Consortium (OGC): OGC 17-069r3: **OGC API - Features - Part 1: Core** [online]. Edited by C. Portele, P. Vretanos, C. Heazel. Available at <http://www.opengis.net/doc/IS/ogcapi-features-1/1.0>
- OGC: [Draft OGC API - Features - Part 2: Coordinate Reference Systems by Reference](http://docs.opengeospatial.org/DRAFTS/18-058.html) [http://docs.opengeospatial.org/DRAFTS/18-058.html]
- OGC: [Draft OGC API - Features - Part 3: Common Query Language](http://docs.opengeospatial.org/DRAFTS/19-079.html) [http://docs.opengeospatial.org/DRAFTS/19-079.html]
- OGC: [Draft OGC API - Features - Part 4: Simple Transactions](http://docs.opengeospatial.org/DRAFTS/20-002.html) [http://docs.opengeospatial.org/DRAFTS/20-002.html]
- OGC: [Draft OGC API - Processes](https://htmlpreview.github.io/?https://github.com/opengeospatial/wps-rest-binding/blob/master/docs/18-062.html) [https://htmlpreview.github.io/?https://github.com/opengeospatial/wps-rest-binding/blob/master/docs/18-062.html]
- Internet Engineering Task Force (IETF): RFC 7946: **The GeoJSON Format** [online]. Edited by H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub. 2016 [viewed 2020-03-16]. Available at <http://tools.ietf.org/rfc/rfc7946.txt>
- Open Geospatial Consortium (OGC): OGC 10-100r3: **Geography Markup Language (GML) Simple Features Profile** [online]. Edited by L. van den Brink, C. Portele, P. Vretanos. 2012 [viewed 2020-03-16]. Available at http://portal.opengeospatial.org/files/?artifact_id=42729
- Open Geospatial Consortium (OGC): OGC 06-103r4: **OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture** [online]. Edited by John R. Herring. 2011 [viewed 2020-11-05]. Available at https://portal.ogc.org/files/?artifact_id=80428
- Open Geospatial Consortium (OGC): OGC 09-025r2: **OGC® Web Feature Service 2.0 Interface Standard – With Corrigendum** [online]. Edited by Panagiotis (Peter) A. Vretanos. 2014 [viewed 2020-11-05]. Available at <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) [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

• Coordinate operation

A process using a mathematical model, based on a one-to-one relationship, that changes coordinates in a source CRS to coordinates in a target CRS, or that changes coordinates at a source coordinate epoch to coordinates at a target coordinate epoch within the same CRS.

SOURCE: ISO 19111:2019 Referencing by coordinates

• Coordinate conversion

A coordinate operation that changes the coordinates in a source CRS to coordinates in a target CRS based on the same datum.

Note 1 to this entry This does not represent a change to the coordinates of the described feature, but rather a different representation of the same coordinate.

Source: SOURCE: ISO 19111:2019 Referencing by coordinates

Examples

- Change Geographic coordinates (Latitude and Longitude) to Map Projection (Easting and Northing)
- Change units from feet to meters

• Coordinate transformation

A coordinate operation that changes coordinates in a source CRS to coordinates in a target CRS in which the source and target CRS are based on different datums.

SOURCE: ISO 19111:2019 Referencing by coordinates

Examples

- Change from GDA94 to GDA2020
- Change from AMG66 to MGA94

• DGGS RS

'DGGS Reference System' a Reference system using zonal identifiers with structured geometry as described by the UML diagram in [Figure 1](#)

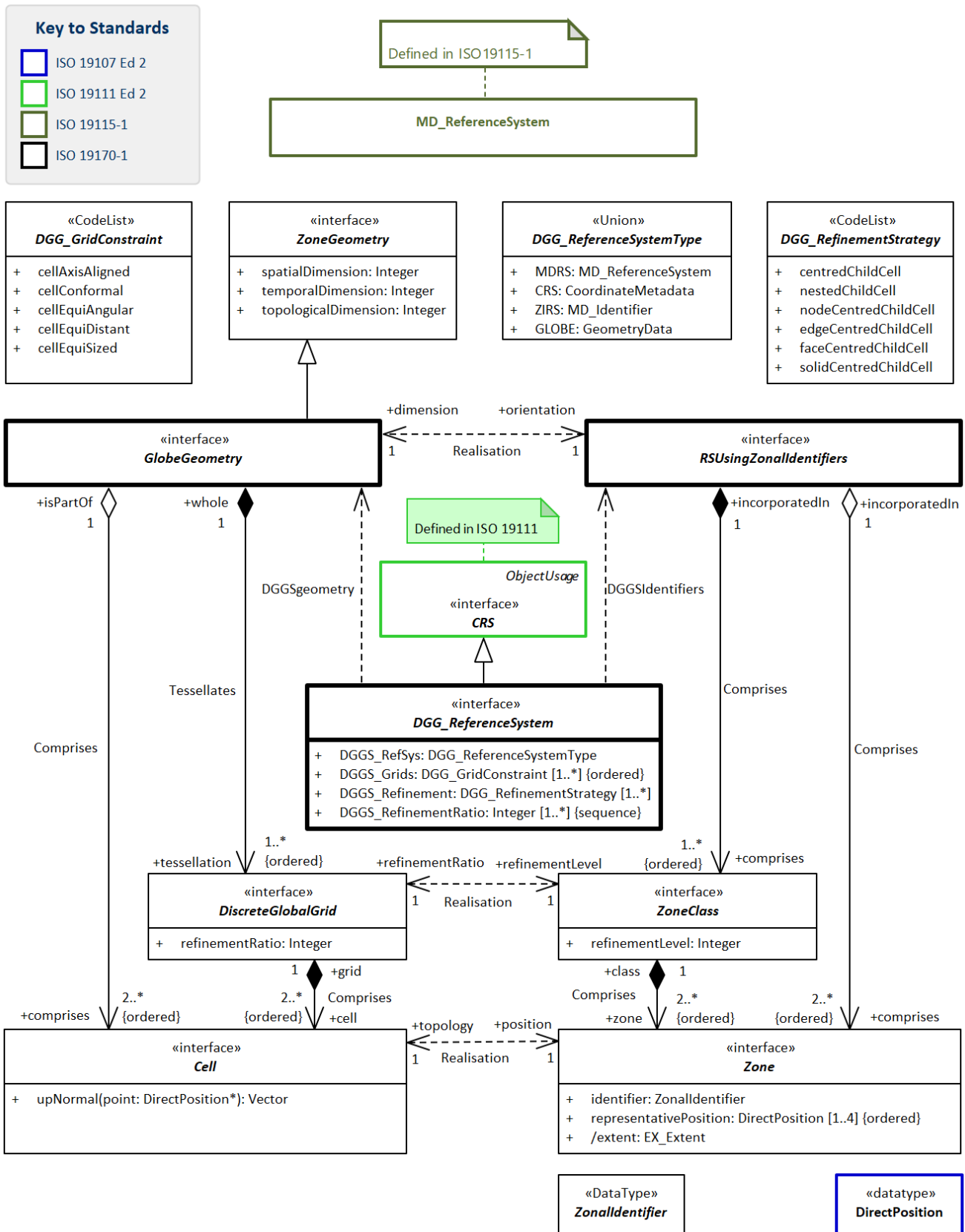


Figure 1. Referencing by zonal identifiers with structured geometry

• **DGGS RS provider (DGGS RS provider service)**

Software library that processes DGGS RS definitions to create zonal identifiers and define their geometry.

Notes

- a DGGs RS provider may be able to be used iteratively to provision a complete DGGs RS.
- a DGGs RS provider service is a DGGs RS provider set up to operate in a web service architecture.
- most DGGs software libraries available today are at the very least DGGs providers
- none of the DGGs provider libraries identified in our stock-take ([Table 1](#)) support DGG_ReferenceSystem or MD_ReferenceSystem. 19170-1 is establishing a template/benchmark for what should happen.

Example

- DGGRID is a DGGs RS provider that needs to be used iteratively to provision a complete DGGs RS, and
- DGGRID is a command line library that would need further configuration to operate as a DGGs RS provider service.

• DGGs RS navigator (DGGs RS navigator service)

Software library that processes topological queries based solely on 'ZonalIdentifiers' as specified by the 'DGGs Core::ZoneQuery' interface.

Notes

- a DGGs RS navigator service is a DGGs RS navigator set up to operate in a web service architecture.
- *Normal geometry operations:* since a Cell has the behavior of Geometry as specified in ISO 19107, Cell.representativePoint and 'Cell.boundary` would be supported. For example:
 - if A and B are both ZonalIdentifiers then the DGGs RS navigator operation A.contains(B) returns a boolean.
 - Other operations in zoneQuery include all the DE-9IM operators as well as parent, child, sibling, parentOf, childOf, siblingOf and two 1D operators relativePosition and relatePosition.
 - ZoneQuery also supports operations spanning a specified number of refinement levels. So effectively queries involving grandchildren, cousins once removed etc can all be processed by adding a levels parameter to the child, sibling etc.

Examples

- rHEALPix is a DGGs RS navigator as well as a DGGs RS provider.
- H3 is also a DGGs RS navigator as well as a DDGS RS provider.
- DGGRID is not a DGGs RS navigator.
- In both libraries the navigator functions would need a wrapper to comply with the nomenclature and syntax in ZoneQuery.

• DGGs datastore

persistent storage for observation values assigned to zonal identifiers

Notes

- each DGGs RS specifies a single geometry for their zones (cells).
- all geometry types present in non-DGGs systems map onto collections of DGGs zones (cells).
- there are three forms that collections of zones can take: single zone, arrays of zones, and ordered arrays of zones, where the ordering indicates a zone connectivity.
- each element of a collection is a ZonalIdentifier.

• DGGs RDF datastore

storage uses ZonalIdentifiers as subject or object to represent a region of space-time and RDF predicates to associate the ZonalIdentifier(s) with observation values

• DGGs array datastore

observation values are stored using arrays in which the array is sorted by ZonalIdentifier, and may therefore define a function that relates ZonalArray offsets to array index

• DGGs raster datastore

observation values are stored as an image type sorted by lat, long (eg geotiff) and a function is defined to relate ZonalIdentifier with d/d-lat, d/d-long zone spacing

• DGGs analytics system

service providing spatial analysis of observations stored in a DGGs datastore

• DGGs quantization service

import process for converting observation data from non-DGGs format to DGGs format

Note

See OGC Topic 21 v2.0 Part 1 standard for detail

• DGGs Query/Broadcast Service

export process of converting DGGs format data to non-DGGs format

Note

See OGC Topic 21 v2.0 Part 1 standard for detail

• DGGs API

APIs to implement each of the above.

• trie

ordered tree data structure used to store a dynamic set or associative array where the keys are usually strings.

Note

Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated; i.e., the value of the key is distributed across the structure. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Keys tend to be associated with leaves, though some inner nodes may correspond to keys of interest. Hence, keys

are not necessarily associated with every node. For the space-optimized presentation of prefix tree, see compact prefix tree.

Note

See <https://en.wikipedia.org/wiki/Trie>

- **zone**

region of space-time

Note

In DGGs the zone is the fundamental unit of space-time. Each zone has a unique zonal identifier (ZoneID), and the zonal identifier has a defined position in a base CRS. The zone has geometry represented by a cell, and like all geometry, the cell has topology. Best practice is for the zonal identifier to be an encapsulation of both position and topology. The distinction between cell and zone was introduced in OGC Topic 21 v2.0 Part 1 both to encompass any spatio-temporal dimensionality, including 2D, 2D+Time, 3D & 3D+Time and to distinguish between the region and its geometry. See OGC Topic 21 v2.0 Part 1 standard for more detail.

Note

cell and cell ID, are only used in this document when referring to specific objects, classes or functions in existing software implementations.

4.1. Abbreviated terms

- API Application Programming Interface
- AusPIX Australian rHEALPix Reference System
- CFP Testbed-16 Call For Participation
- DAPA Testbed-16 Data Access and Processing API for Geospatial Data
- DGGs Discrete Global Grid System
- GIS Geographic Information System
- H3 Uber's Hexagonal Hierarchical Spatial Index
- HDF5 Hierarchical Data Format v5
- JSON JavaScript Object Notation
- OLAP Online analytical Processing
- rHEALPix rearranged Hierarchical Equal Area iso-Latitudinal Pixelisation
- TB-16 Testbed-16
- TB16Pix Testbed-16 rHEALPix Reference System
- URI Universal Resource Identifier
- UTM Universal Transverse Mercator

Chapter 5. Overview

Preface - provides the required Preface information such as copyright, warnings and license agreement.

Subject - introduces DGGs in the context of the TB-16 experiment.

Executive Summary - provides the Executive Summary of this report.

References - lists all normative references relevant to the outcomes of this experiment.

Terms and definitions - lists the key terms and definitions relevant to this report.

Overview - (this document) provides an overview and extended Table of Contents of this report.

DGGs and DGGs Reference System Selection - discusses the key criteria and justification for the selection of the DGGs libraries and associated DGGs Reference Systems used in this experiment.

DGGs API - discusses the context and API options for the DGGs thread and the definitions of the Features and Processes APIs that have been deployed.

Use Cases - presents and discusses the Use Cases employed to demonstrate the value of DGGs in the context of this experiment.

DGGs Server and API - presents and discusses the implementation of the DGGs Server, OGC API - Features endpoint, and the preparation of the data served for this experiment.

DGGs Demo Client - presents and discusses the implementation of the DGGs Client used in this experiment.

DGGs Enabled Data Services - presents and discusses the implementation of the DGGs-enabled Data Server, OGC API - Processes endpoint, and the preparation of the data served for this experiment.

Future Tasks - discusses Future Tasks and Activities that need to be undertaken to support the standardized implementation of DGGs and services that implement the DGGs API.

Chapter 6. DGGs and DGGs Reference System Selection

6.1. Semantics for DGGs libraries and their association with the current DGGs draft standards

The new draft DGGs Abstract Specification (OGC Topic 21 v2.0 / ISO/DIS 19170-1) defines 1.) a DGGs Reference system as 'referencing by zonal identifiers with structured geometry', and 2.) a DGGs as a holistic system comprising:

- A DGGs Reference system (DGGs RS);
- A suite of DGGs Functions for:
 - a. Quantizing data against the DGGs RS,
 - b. Querying the topology of zones in a DGGs RS, and
 - c. Interoperating with traditional W*S and OGC API systems.

Most existing DGGs libraries pre-date the new draft Abstract Specification (AS) and as a consequence they only partially fulfil the requirements of the draft AS. Furthermore, they are not structured neatly into modules that follow the above pattern. This has created confusion as to what comprises a DGGs. As such participants in this thread held diverse views of what constituted a DGGs. As the diversity was explored the need for some additional concepts or terms that reflected the different roles that a library might play in DGGs were identified. The participants also identified that some existing libraries supported a single DGGs RS, while others supported whole families of DGGs RSs. An analogy might be the difference between a library that locked in the Universal Transverse Mercator (UTM) Zones 3 projection as compared to a library that supports Transverse Mercator projections. With appropriate configuration parameters the library can support any UTM Zone - including UTM Zone 3. Following the structure outlined above, the participants distinguished between libraries that were DGGs RS *providers* - and those that were DGGs RS *navigators*.

6.1.1. DGGs RS provider

Providers support either a single DGGs RS, or configuration tools that can be used to support a family of similar DGGs RSs. For the DGGs RSs that they support, *providers* can generate ZoneIds and their associated geometry.

6.1.2. DGGs RS navigator

Navigators support the topological queries on zones based on ZoneIds. These are the usual [Dimensionally Extended 9-Intersection Model](https://en.wikipedia.org/wiki/DE-9IM) [https://en.wikipedia.org/wiki/DE-9IM] (DE-9IM) functions such as *within*, *overlap*, *contains* etc.

6.2. The question of DGGs data

Early in the discussion between participants, the question was raised as to whether there was such a thing as DGGs data, and if there were, what would a DGGs datafile look like? For anybody coming into DGGs from a traditional GIS background, and thinking in terms of DGGs as a reference system, the obvious answer was no. For example, concepts such as World Geodetic System 1984 (WGS84) data or UTM 3 data have very little meaning. On the other hand, those participants heavily involved in DGGs thinking were convinced that DGGs data was a very real thing. Thinking about spatial data is typically in terms of its storage, and storage design is intimately bound to the form of the geometry being stored — Shapefiles, HDF5, GeoJSON, LASer (LAS) formatted files — are all solutions to the needs of storing geometries. Traditional reference systems are independent of geometry, and as a consequence the choice of reference systems is independent of the choice of data format.

By contrast, DGGs reference systems define their own geometry. This geometry is implied by the ZoneID. So, within a DGGs a geometry does not need to be explicitly provided as coordinates, and therefore does not need to be encoded in a traditional spatial data format. Instead, DGGs data only needs to record the ZoneID(s) associated with the feature, observation, or raster data zone (cell). This is very much like any other unique identifier associated with a list of attributes, and so DGGs data can be stored in many typical datastores. What makes it spatial data is the presence of the DGGs RS's navigator functions that perform spatial topology operations using the ZonalID(s) in lieu of coordinate geometry. As a consequence, explicit read/write support for DGGs data in a DGGs library is not required. What is expected is handling of the quantization roles that are defined by the draft OGC DGGs Abstract Specification. These roles tell us the geometric relationship between a record and a ZoneID. They include a single coordinate (the zone centroid), or an area (enclosed by the zone's boundary), or a tile.

6.3. DGGs libraries

A stock-take of DGGs libraries was performed and the following are the libraries found.

Table 1. DGGs Libraries

Name	Author	Cell geom	Docs	Repo	Lang. Bind	License	DGGs RS prov.*	Quant.*	DGGs RS nav.*	Interop.*
DGGrid	Kevin Sahr [mailto:sahrk@sou.edu], Southern Oregon University [https://sou.org]	hex or tri	STCL [https://www.discreteglobalgrids.org/]	DGGrid on github [https://github.com/sahrk/DGGRID]	C++ cmd-line	AGPL 3.0 [https://github.com/sahrk/DGGRID/blob/master/LICENSE]	Y (config)	N	N	N

Name	Author	Cell geom	Docs	Repo	Lang. Bind	License	DGGS RS prov.*	Quant.*	DGGS RS nav.*	Interop.*
DGGri dR (wrapper to use DGGri d in R)	Richard Barnes [mailto:rbarnes@umn.edu]	hex or tri	STCL [https://www.discretglobalgrids.org/]	DGGri dR on github [https://github.com/r-barnes/dggridR]	R, C++	MIT [https://github.com/r-barnes/dggridR/blob/master/LICENSE]	Y (config)	Y	N	N
H3 (built on DGGri d)	Uber	hex	Uber's H3 index [https://eng.uber.com/h3/]	H3 on github [https://github.com/uber/h3]	C, java	Apache 2.0 [https://github.com/uber/h3/blob/master/LICENSE]	Y	N	Y	N
OpenE AGGR	DSTL(UK) & RiskAware	tri or hex	Lit, Prog, Software [https://github.com/riskaware-ltd/open-eaggr/tree/master/Documents]	OpenEAGGR on github [https://github.com/riskaware-ltd/open-eaggr]	C	LGPL v3 [https://github.com/riskaware-ltd/open-eaggr/blob/master/COPYING.LESSER]	Y (config)	N	N	N
rHEAL Pix	Robert Gibb [mailto:gibbr@landcareresearch.co.nz] Manaaki Whenua [https://www.landcareresearch.co.nz]	quad	Original mathsdataset/rhealpix-discrete-global-grid-system]	rhealpix-ddgs-py on github [https://github.com/ManaakiWhenua/rhealpix-ddgs-py]	python	CC-BY 4.0 [https://creativecommons.org/licenses/by/4.0/] & LPGL for original code	Y (config)	N	Y	N

Name	Author	Cell geom	Docs	Repo	Lang. Bind	License	DGGS RS prov.*	Quant.*	DGGS RS nav.*	Interop.*
AUSPix (implementation of rHEAL Pix)	GA & CSIRO	quad	Loc-I [http://locationindex.org/home.html] & AusPIX [https://locationindex.s3-ap-southeast-2.amazonaws.com/Implementation++DGGS+v0.1.pdf]	AusPIX on github [https://github.com/GeoscienceAustralia/AusPIX_DGGS]	python	CC-BY 4.0 [https://creativecommons.org/licenses/by/4.0/] & LPGL for original code	Y	Y	Y	N
PYXIS	Perry Peterson Global Grid Systems	hex		n/a		commercial product	Y	Y	Y	Y
TerraNexus	Matthew Purss, Pangaea Innovations Pty. Ltd.	quad & tri		n/a	python	commercial product	Y	Y	Y	Y
	... (add others here)									

NOTE

* None of the libraries implements complete functionality in any of the four categories. Therefore the use of 'Y' in these columns indicates partial fulfilment. However, for the proposed purposes of TB-16 DGGS thread any gaps are trivial.

6.4. DGGS Libraries selected by the Testbed-16 DGGS thread

To demonstrate that the DGGS work undertaken was applicable to multiple DGGS, the decision was made to implement two DGGS, and the libraries chosen were H3 and rHEALPix.

- **H3:** Use of H3 is resulting in the development of a significant user community and as a consequence H3 is the most mature in the sense of formal releases and bindings.
- **rHEALPix:** Being used as AusPIX by the Australian Government's Loc-I project. Also TB-16 thread participants were be able to pick up some of the additional code developed for AusPIX.

6.5. DGGS Reference systems selected by the Testbed-16 DGGS thread

H3 only supports one DGGS RS, so there are no further parameter choices to be made. The H3's reference system is referred to as H3RS.

rHEALPix provides a number of hard-coded references systems, such as for covering spherical vs ellipsoidal earth models. For example, AusPix uses the default hard-coded WGS84 ellipsoidal reference system based on the zero meridian. Since Testbed-16 is developing solutions that may be demonstrated in a number of continents, the decision was made to choose a rotated version of the WGS84 that places all the corners of the initial cube in the sea. This reduces the area of land covered by the most distorted zones. This reference system as TB16Pix.

The code to provide TB16Pix, print the definition and export zone centers and edges for visualization in Google Earth is:

```

from rhealpix_dggs.projection_wrapper import *
from rhealpix_dggs.dggs import *
from rhealpix_dggs.ellipsoids import *

# define our WGS84 ellipsoid rotated so that all the corners of the cube lie in water
WGS84_TB16 = Ellipsoid(a=6378137.0, b=6356752.314140356, e=0.0578063088401, f
=0.003352810681182, lon_0=-131.25)

# create our DGGs RS based on the defined ellipsoid
TB16Pix = RHEALPixDGGs(ellipsoid=WGS84_TB16, north_square=0, south_square=0, N_side=3)

# fyi print out the definition in full
print(TB16Pix)

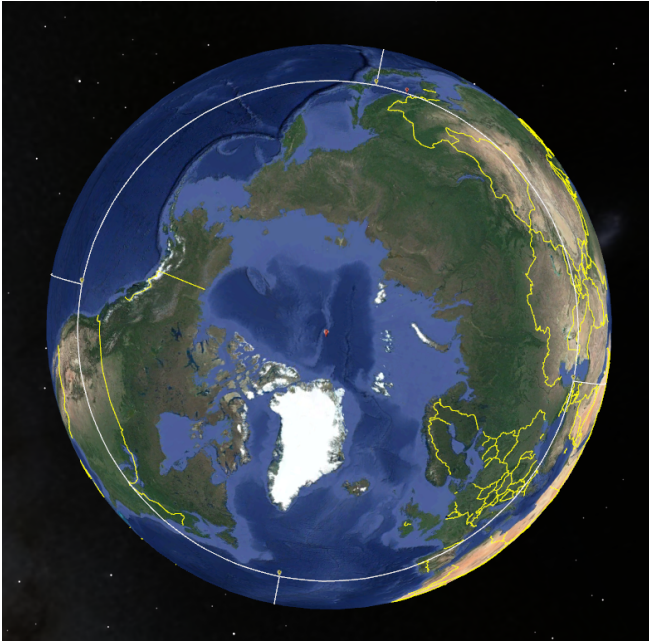
# fyi print out the coordinates of the corners and centres of the cube's faces,
# note that the rHEALPix code doesn't support direct query of the top level zones,
# so appropriate children are used
c = TB16Pix.cell(('O', 0)); print(my_round(c.nw_vertex(plane=False),9)) # Caspian Sea
c = TB16Pix.cell(('P', 0)); print(my_round(c.nw_vertex(plane=False),9)) # Sea of Japan
c = TB16Pix.cell(('Q', 0)); print(my_round(c.nw_vertex(plane=False),9)) # NW Pacific
Coast
c = TB16Pix.cell(('R', 0)); print(my_round(c.nw_vertex(plane=False),9)) # Nth Atlantic
c = TB16Pix.cell(('O', 6)); print(my_round(c.vertices(plane=False)[3],9)) # Indian
Ocean
c = TB16Pix.cell(('P', 6)); print(my_round(c.vertices(plane=False)[3],9)) # Australian
Bight
c = TB16Pix.cell(('Q', 6)); print(my_round(c.vertices(plane=False)[3],9)) # Sth
Pacific
c = TB16Pix.cell(('R', 6)); print(my_round(c.vertices(plane=False)[3],9)) # Sth
Atlantic
c = TB16Pix.cell(('N', 4)); print(my_round(c.nucleus(plane=False),3)) # N - Nth Polar
face
c = TB16Pix.cell(('O', 4)); print(my_round(c.nucleus(plane=False),3)) # O - Oriental
face
c = TB16Pix.cell(('P', 4)); print(my_round(c.nucleus(plane=False),3)) # P - Pacific
face
c = TB16Pix.cell(('Q', 4)); print(my_round(c.nucleus(plane=False),3)) # Q - Americas
face
c = TB16Pix.cell(('R', 4)); print(my_round(c.nucleus(plane=False),3)) # R - Africa
face
c = TB16Pix.cell(('S', 4)); print(my_round(c.nucleus(plane=False),3)) # S - Sth Polar
face

```

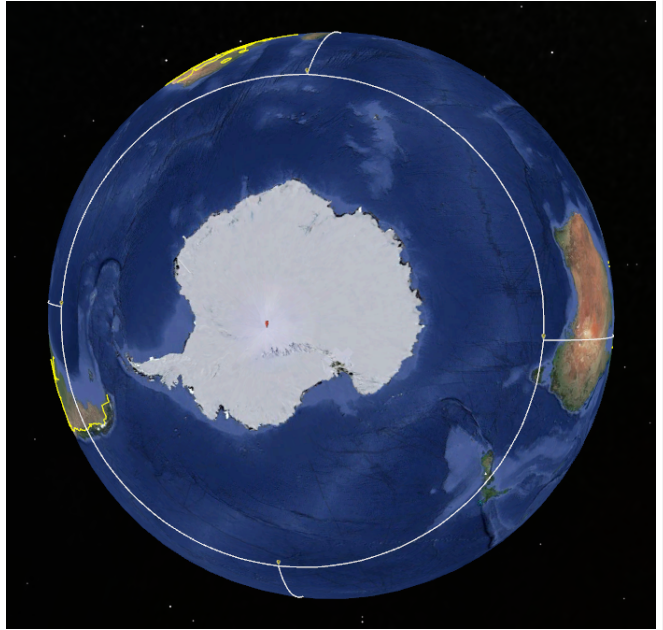
As illustration the corners of the northern and southern zones are shown in the next two figures:

Table 2. TB16Pix Polar zones showing corners in water

TB16Pix Zone N corners



TB16Pix Zone S corners



Chapter 7. DGGS API

7.1. Options for DGGS and OGC API

The participants had considerable discussions about the context in which an API that talked DGGS would be used. A couple of issues were central to the discussion:

- DGGS RS provides geometry, and the geometry is encoded in the ZoneID. So, a true DGGS API does not need to exchange geometry in the form of coordinates, whereas OGC API - Features is built on the presumption that geometry will always be exchanged.
- DGGS RS provide a series of successively finer Zone resolutions. Each resolution represents a defined level of known spatial precision. However, OGC API - Features has no concept of spatial precision, so some form of automatic assignment of spatial precision may be required.

The alternative contexts discussed were:

- A DGGS client calling a DGGS server — both understand what a ZoneID represents:
 - Client and server using the same DGGS RS — no ZoneID conversion required.
 - Client and server using different DGGS RS — ZoneID conversion or negotiation required.
- A DGGS client calling an OGC API - Feature server or an OGC API - Processes server:
 - DGGS client understands how to generate a ZoneID's coordinate geometry for the API — Server does not need to understand DGGS ZoneID.
 - Server understands how to interpret ZoneID — Client does not need to generate a ZoneID's geometry.

There was also discussion about whether to implement an API that conforms to OGC API - Features or OGC API - Process. While OGC API - Features would be useful for allowing people to browse a DGGS and has clarity of meaning, it would also potentially be very chatty in use for analytics (i.e. a potentially very large amount of data, in the form of JSON documents, being transferred either as very large OGC API - Features collection documents, or many transactions of individual OGC API - Features documents for each feature of a collection). By comparison OGC API - Processes is relatively opaque, but fits very well with the DGGS premise that DGGS are particularly suited to Big Data applications where processing should be passed to the data. This is as opposed to the case in which features are passed to the client for processing.

7.2. Aligning DGGS to OGC API - Features

Intended content

NOTE

This section will discuss the way that DGGS map onto collections and features in OGC API - Features.

7.3. Aligning DGGS to OGC API - Process

For an API that is a profile of the draft OGC API - Process, the basic resources "are" Processes. Any

particular API and API implementation offers particular processes or process types, organizes them in characteristic structures, and supports returning particular representations. The most common representations will be 1) descriptions of the processes, or 2) the result of a process "job" invoked by a request.

What really makes an API, though, is how those resources are organized. The trend with OGC API - Processes (via GET operations) is to group processes according to the primary dataset that they act on, what might be termed the process "affordance". This matches up well with the draft OGC API - Common pattern of dividing resources into collections. Depending on whether a hierarchical organization is supported, each collection is either a collection of collections, or a dataset composed of data elements. The sleight of hand, though, is that a dataset in a Processes API is not a target resource, but a means of organizing processes that are able to act on it.

What makes sense is that a DGGs be thought of as a dataset, with or without data properties, that organizes retrieval, linking, and processing types. The dual nature of a DGGs as both a processing engine and a dataset/collection of geometry objects creates flexibility in the way an OGC API for DGGs could be structured as:

- system first (e.g. `/ogcapi/dggs/{DGGs_RS_ID}/processes/...`, `/ogcapi/dggs/{DGGs_RS_ID}/collections/...`, etc...), or
- data first (e.g. `/ogcapi/collections/{collectionID}/dggs/{DGGs_RS_ID}/...`, `/ogcapi/processes/{processID}/dggs/{DGGs_RS_ID}/...`, etc...).

These concepts require more elaboration during subsequent OGC standards development and/or OGC Innovation Program activities to standardize the concept of an OGC API for DGGs.

7.4. Implementation in the deliverables

As a result of these discussions the decision was made to implement DGGs in the context of two distinct APIs.

- **D139** an OGC API - Features oriented API hosted by a DGGs Server that could support both a native DGGs client, in this case the **D138** Demo Client, and an OGC API - Features client that needs the geometry to be provided. The server would provide DGGs Zones as Features without data. The server would deliver some DGGs data layers, delivered either as DGGs Zones using their ZonalIds or as traditional Features using each zone's coordinate geometry.
- **D137** an OGC API - Processes server would be built inside GeoServer. The API instance would perform a selection of ZoneQuery operations on DGGs Zones.

7.5. Treatment of geometry

Acknowledging the conundrum of how to deliver geometry, the following strategy is being used by both API implementations:

- DGGs RS were treated as if they were a collection of feature datasets, with each level of the DGGs zone hierarchy corresponding to an item in the collection. This approach provided an elegant solution to the association of geometry with a ZoneId.

- While each DGGS RS is presented as a (collection of) datasets, in the first instance the datasets were generated on the fly as required. This was achieved by embedding the library in the server.
- The client chooses how to receive geometry by specifying its desired protocol. For example, specifying GeoJSON results in geometry presented as traditional coordinates according to the GeoJSON specification, while specifying JSON results in geometry being passed encoded in the ZoneID on the presumption that the client does not need the coordinates.
- Both the OGC Features and Processes APIs were designed to support all the functions specified by ZoneQuery (c.f. Table 3). However only a subset was implemented in this Testbed. In the OGC API - Features implementation, the default representation of each Zone included reference to parents, siblings and children - all generated on-the-fly. The OGC API - Processes implementation offered those same functions as processes.

7.6. Operations specified for ZoneQuery

These are the operations defined in OGC Topic 21 Part 1 v2.0 and ISO 19170-1 Core for ZoneQuery.

Table 3. Elements of Core Query Functions::ZoneQuery class

Name:	ZoneQuery					
Definition:	ZoneQuery redefines the DE-9IM operations in Query2D, Query3D and provides relativePosition and relatePosition operations for the topology of zones.					
Stereotype:	Interface					
Abstract:	true					
Associations:	(none)					
Public attributes:	<i>Name</i>	<i>Definition</i>	<i>Derived</i>	<i>Obligation</i>	<i>Maximum occurrence</i>	<i>Data type</i>
	boundary	boundary of the combined spatial geometries of the zones in the query	true	M	1	Geometry
	boundaryType	boundary type of the combined spatial geometries of the zones in the query		M	1	BoundaryType
	convexHull	convex hull of the combined spatial geometries of the zones in the query	true	M	1	Geometry

Name:	ZoneQuery			
Operations:	Name	Parameters:ParameterType	Return type	Definition
	distance	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Distance	A.distance(B)
<<query>> (1D)	relativePosition	(another:ZonalIdentifier, projectTo:DirectPosition[4])	RelativePosition	A.relativePosition(B,(0,0,0,1))
<<query>>	contains	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.contains(B) \square $A \supset B$
	crosses	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.crosses(B)
	disjoint	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.disjoint(B) \square $A \cap B = 0$
	equals	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.equals(B) \square $A = B$
	intersects	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.intersects(B) \square $A \cap B \neq 0$
	overlaps	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.overlaps(B)
	touches	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.touches(B)
	within	(another:ZonalIdentifier, projectTo:DirectPosition[4])	Boolean	A.within(B) \square B.contains(A)
	withinDistance	(another:ZonalIdentifier, dist:Distance, projectTo:DirectPosition[4])	Boolean	A.withinDistance(B) \square A.distance(B) < dist
	parentOf	(another:ZonalIdentifier, inheritID:Boolean, projectTo:DirectPosition[4])	Boolean	A.parentOf(B)
	childOf	(another:ZonalIdentifier, inheritID:Boolean, projectTo:DirectPosition[4])	Boolean	A.childOf(B)
	siblingOf	(another:ZonalIdentifier, inheritID:Boolean, projectTo:DirectPosition[4])	Boolean	A.siblingOf(B)

Name:	ZoneQuery			
<<set>>	buffer	(dist:Distance, projectTo:DirectPosition[4])	ZonalIdentifier	A.buffer(dist)
	difference	(another:ZonalIdentifier, rangeRefine:refinementLevelRange, projectTo:DirectPosition[4])	ZonalIdentifier	A.difference(B) \square A-B
	intersection	(another:ZonalIdentifier, rangeRefine:refinementLevelRange, projectTo:DirectPosition[4])	ZonalIdentifier	A.intersection(B) \square $A \cap B$
	symDifference	(another:ZonalIdentifier, rangeRefine:refinementLevelRange, projectTo:DirectPosition[4])	ZonalIdentifier	A.symDifference(B) \square (A-B) \square (B-A)
	union	(another:ZonalIdentifier, rangeRefine:refinementLevelRange, projectTo:DirectPosition[4])	ZonalIdentifier	A.union(B) \square $A \cup B$
	parent	(inheritID:Boolean, levels:Integer, projectTo:DirectPosition[4])	ZonalIdentifier	A.parent(B)
	child	(inheritID:Boolean, levels:Integer, projectTo:DirectPosition[4])	ZonalIdentifier	A.child(B)
	sibling	(inheritID:Boolean, levels:Integer, projectTo:DirectPosition[4])	ZonalIdentifier	A.sibling(B)
<<reference>> (1D)	relatePosition	(another:ZonalIdentifier, relate:RelativePosition, projectTo:DirectPosition[4])	Boolean	A.relatePosition(B, enum,(0,0,1,0))
<<reference>>	relate	(another:ZonalIdentifier, matrix:CharacterString, projectTo:DirectPosition[4])	Boolean	A.relate(B,matrix)
Constraints:	(none)			

7.7. OGC API - Features for DGGs description

Addressing the Call for Participation (CFP) clause *D139 DGGs Enabled Data Services*, the OGC API - Features implementations for DGGs datasets should support:

1. "Cell IDs as spatial filters"

2. "Converting Cell IDs to geographic filters"
3. "Cell ID(s) to geographic location conversion"

The specifics of these implementations are documented in the next section.

7.7.1. API Features implementation

A new OGC API framework was created for this project called *OGC LD API*. This is based on the fact that the framework uses Linked Data mechanics to provide both OGC API and other API functionality. The implementations based on this framework pass many of the OGC API - Features tests (*Abstract Tests* and *Requirements*) with the intention being that all tests will pass in future versions of the framework.

To test the implementations against the OGC API - Features *Abstract Tests* and *Requirements*, a testing client was also developed. For this Testbed, the client was called the *OGC API LD Test Client* ("the *TC*"). This client implemented not only the *Abstract Tests* and *Requirements* of OGC API - Features but also tested lists in the *Content Negotiation by Profile* specification for Linked Data APIs [1].

So far, the API framework only demonstrates delivering TB16Pix DGGs data. This is due to rather than multiple DGGs the focus was on the creation of 2+ APIs of content for the same DGGs for a DGGs client to consume.

7.7.1.1. Endpoints

Table 4. Endpoint summary

Code	Description
Capabilities	
/	landing page
rootGet	
/conformance	information about specifications that this API conforms to
/collections	the list of supported collections
DGGS Access	
/collections/{collectionId}	Describes a particular Feature Collection
/collections/{collectionId}/items	Access the list of Features within a Collection. Can list either all the Features, or a particular subset based on a, WGS84 bbox, a coarse DGGS CellID (equivalent to a bbox), or a DGGS quadrilateral specified by two DGGS CellIDs (equivalent to a bbox)
/collections/{collectionId}/items/{itemId}	Access the definition of a particular Feature
Content Negotiation	

{API|Collection|Items
List|Feature
URI}?_profile=alt

Access the list of *Profiles* (model views) and *Formats* (media types) available for the API system, a Collection, an Item List or a Feature. This is the standard *Content Negotiation by Profile* [1] mechanics

7.7.1.2. Feature/Geometry association and representation

OGC APIs make a Feature/Geometry association whereby a *Feature* may have more than one *Geometry*. The APIs implemented here list features with multiple DGGS and non-DGGS geometries.

The OGC API specification [O AFC] "does not mandate a specific encoding or format for representing features or feature collections", however mandates the data model and responses to requests for each *Feature* and requires that the [GeoJSON] specification be used to format it. GeoJSON itself is locked into using the WGS84 Coordinate Reference System (CRS) making it unsuitable for DGGS geometries. The OGC API specification gives an example of a GeoJSON response for a feature which is reproduced here:

```
{
  "type" : "Feature",
  "links" : [ {
    "href" : "http://data.example.com/collections/buildings/items/123?f=json",
    "rel" : "self",
    "type" : "application/geo+json",
    "title" : "this document"
  },
  ...
  , {
    "href" : "http://data.example.com/collections/buildings",
    "rel" : "collection",
    "type" : "application/json",
    "title" : "the collection document"
  } ],
  "id" : "123",
  "geometry" : {
    "type" : "Polygon",
    "coordinates" : [ ... ]
  },
  "properties" : {
    "function" : "residential",
    "floors" : "2",
    "lastUpdate" : "2015-08-01T12:34:56Z"
  }
}
```

The code example above is a partial reproduction of *Example 13*, from [O AFC].

The feature encoding in the example above contains links to related API endpoints, a geometry encoding, and several property key/value pairs. While this form of encoding is relatively flexible, it cannot natively communicate DGGS geometries. This is because all GeoJSON geometries are ordered lists of coordinate pairs (longitude & latitude or x & y etc.) that must be expressed using the

WGS84 Coordinate Reference System (CRS). DGGs geometries require coordinates to be represented as lists of Cell ID and are a different CRS to WGS84.

To enable DGGs geometry encoding in a JSON format similar to GeoJSON, both the particular DGGs must be indicated and lists of Cell IDs must be used in place of coordinate pairs. Instead of the following GeoJSON:

```
{
  ...
  "id" : "123",
  "geometry" : {
    "type" : "Polygon",
    "coordinates" : [ [Long_1, Lat_1], [Long_2, Lat_2], [Long_3, Lat_3] ... [Long_N,
Lat_N] ]
  },
  ...
}
```

We would have:

```
{
  ...
  "id" : "123",
  "geometry" : {
    "type" : "Polygon",
    "dgg": "https://w3id.org/dggs/tb16pix",
    "coordinates" : [[ CellID_1, CellID_2, CellID_3... CellID_N ]]
  },
  ...
}
```

Above the URI <https://w3id.org/dggs/tb16pix> for the property "dgg" indicates that the TB16Pix DGGs RS is used.

This GeoJSON-like format for DGGs is trivial to specify but, currently, will not validate as GeoJSON. Thus a specification update or a separate specification is necessary for this modified format to be supported. If a specification update was selected, the property "dgg" could easily be generalised to "crs" to allow the GeoJSON update to use many CRSs/DGGs.

7.7.1.3. DGGs geometry types

In GeoJSON - and many other common spatial data systems - a geometry's *type* is something like POINT, POLYGON, LINESTRING etc. These *types* give an indication of the role played by the *Geometry* such as the *Feature* the geometry is a property of as well as the structure of the geometry literal. In other words, is it a single coordinate point, a list of points or a closed ring etc.

GeoJSON supports the following geometry types:

- Point
- MultiPoint
- LineString
- MultiLineString
- Polygon
- MultiPolygon

These geometry types do not convey the full extent of known non-DGGS spatial geometries. For instance grids, triangular irregular networks, and so on. They also do not necessarily correspond to useful or more powerful geometry types within DGGSs.

DGGS *native* geometries might consist of geometry types such as:

- OrdinateList - Cell Ids representing a set of points
 - Similar to a Point or MultiPoint
- DirectedOrdinateList - Cell Ids ordered by connectivity representing a sequence of points
 - Similar to a LineString
- CellList - Similar to a Polygon or MultiPolygon, noting that a CellID

While these DGGS *native* geometries are similar to regular geometry types, there is a requirement for each to communicate quantization decisions (e.g. resolution and whether compression is used). Further many of the rules for DGGS shape creation are different to regular geometries. For example, within a *DirectedTileList*, connectivity is assumed to be by ‘straight’ vectors. This is unlike anything in non-DGGS concepts.

The geometry types used throughout the remainder of this chapter for DGGS literal values are implemented for maximum interoperability with regular, non-DGGS, geometry literals and spatial data models such as GeoSPARQL. Future OGC APIs might implement *native* geometry types or perhaps native and non-native types.

7.7.1.4. Further GeoJSON limitations

In addition to the CRS and the form of the geometry location values (*coordinates* in WGS84 but only *ordinates* in a DGGS since DGGS Cell IDs are single items, not pairs of values), GeoJSON is not able to directly express *semantic* properties for features. Properties for GeoJSON Features other than geometries are just listed as key/value pairs, such as "*floors*" : "2", etc. in the first of the two examples above. A *semantic* representation of this property would define what *floor* means, what datatype 2 is and so on. This does not present an issue for the geometry literal itself (the Cell IDs) but does prevent the use of GeoJSON to communicate with no ambiguity values for Features .

Data specifications such RDF [2] are designed to convey semantics by defining relationships, datatype and so on. RDF can be serialized in JSON-LD ^[10] and thus data similar to GeoJSON can be produced. The following example communicates that Feature 1 has a *floors* property with a value of 2 but, unlike the example three above, it uses JSON-LD not GeoJSON and thus it:

- Identifies the Feature with a universally unique and resolvable ID, an HTTP URI.

- Defines the property *floors* by quoting an identity for it that leads to an ontology definition.
- Here the dummy URI <http://example.com/some-ontology/floors> allows it to be contrasted with the GeoJSON example's literal of "floors".
- Fixes a datatype for the value "2".
- Here it is an XML integer, as defined by <http://www.w3.org/2001/XMLSchema#integer>.

```

---
[
  {
    "@id": "http://example.com/feature/1",
    "@type": [
      "http://www.opengis.net/ont/geosparql#Feature"
    ],
    "http://example.com/some-ontology/floors": [
      {
        "@type": "http://www.w3.org/2001/XMLSchema#integer",
        "@value": "2"
      }
    ]
  }
]
---

```

A "downscaling" of JSON-LD to GeoJSON is possible if the client is "happy" with the ambiguities of GeoJSON.

7.7.1.5. Geometry Formats other than GeoJSON

In addition to GeoJSON representations of feature information, a Well Known Text (WKT) like format is presented by the APIs. This appears in the geometry literal values for features as communicated by the "geosp" (GeoSPARQL) view (profile) supported by the API. The following RDF data communicates the geometry of a Feature, *Statistical Area 80101100105* in both normal WKT and WKT-DGGS terms:


```

@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix geox: <https://linked.data.gov.au/def/geox#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<https://linked.data.gov.au/dataset/asgs2016/statisticalarealevel1/80101100105> a
geo:Feature ;
  geo:hasGeometry [
    geo:asWKT "POLYGON ((
      149.075478002 -35.260227984,
      149.075592138 -35.260032958,
      ...
      149.075478002 -35.260227984))"^^geo:WktLiteral ;
  ],
  [
    geo:asWKT "<https://w3id.org/dggs/tb16pix>
      POLYGON (P67303515562
        P67303515565
        ...
        P67303523664)"^^geox:wktDGGSLiteral ;
  ]
.

```

In the code above, linebreaks and tabspace have been added to make the content more readable. Ellipses indicate where coordinate values have been removed for brevity.

The WKT DGGs literal value above is indicated in the same way as a regular WKT value, with the `asWKT` property of the `Feature` but then the WKT DGGs literal is typed `geox:wktDGGSLiteral` which is a sub property of the standard `geo:wktLiteral` in the GeoSPARQL standard [3]. This new literal type, `geox:wktDGGSLiteral` is currently defined in the GeoSPARQL Extensions Ontology (see section [DGGs Enabled Data Services](#)) but it is hoped that it will be absorbed into the GeoSPARQL standard with an update to it which is currently underway ^[11].

7.7.1.6. Geometry validation tooling

Since the formats for DGGs geometry literals are not standardized yet, the testbed participants developed a stand-alone DGGs geometry validator tool which would be able to be used to validate DGGs literals of the sorts mentioned in the above two sections. This tool is a Python library and is based on an existing WKT validator and is called DGGs GV (Geometry Validator). The tool is listed in the section [DGGs Enabled Data Services](#).

For the WKT DGGs format described above, the tool uses Extended Backus Naur Form (EBNF) ^[12] descriptions to define the format's requirements. An incomplete example is the following set of example EBNF statements which define how a WKT DGGs polygon-type geometry must be formulated (as shown in the example in the previous section):

```

wkt_dggs = crs geometry ;

crs = left_angle http_uri right_angle ;

left_paren = "(" ;
right_paren = ")" ;
left_angle = "<" ;
right_angle = ">" ;

http_uri = httpaddress | httpsaddress ;
httpaddress = "http://" hostport [ / path ] ;
httpsaddress = "https://" hostport [ / path ] ;

...

geometry =
    point_text_representation |
    curve_text_representation |
    surface_text_representation |
    collection_text_representation;

surface_text_representation =
    curvepolygon_text_representation ;

curvepolygon_text_representation =
    "CURVEPOLYGON" [ z_m ] curvepolygon_text_body |
    polygon_text_representation |
    triangle_text_representation ;

polygon_text_representation =
    "POLYGON" [ z_m ] polygon_text_body;

...

```

The completed form of the above code is being developed in the following Git repository as part of the DGGs GV library:

<https://github.com/surroundaustralia/dggsgv/blob/master/dggsgv/wkt-dggs.ebnf>.

The above format says that a WKT DGGs polygon geometry: - must have a CRS (DGGs) identifier that is an HTTP/HTTPS URI enclosed in "<" & ">", - followed by a space, - then the word "POLYGON", - then a space, - then optionally "Z" or "M" and - then the polygon's coordinates which, for a WKT DGGs geometry are a list of DGGs Cell IDs ordered largest to smallest (in Cell area), separated by spaces and enclosed in "(" & ")".

7.7.1.7. Geometry roles

When establishing the DGGs literal values for features delivered by the OGC API - Features instances, it was clear that the *role* that a geometry played, as separately from its *type*, e.g. point,

polygon etc., is something not described in the current GeoSPARQL ontology or the data formats for the literals (GeoJSON, WKT etc.). The notion of a *role* for a geometry is included in [4] and is proposed for inclusion in GeoSPARQL 1.1.

The suggested geometry types to be supported for DGGs literals and thus able to be validated by DGGs GV are the same as existing WKT types which are:

- (Multi)Point
- (Multi)LineString
- (Multi)Polygon
- GeometryCollection

The roles which geometries might play are not yet fully formalized. Suggested roles are found in organization's vocabularies, such as the Geological Survey of Queensland's *Geometry Role*.^[13] and include roles such as:

- Boundary
 - Bounding box
 - Bounding circle
 - Concave hull
 - Convex hull
- Centroid
- Detailed geometry

From the options above, that the geometry for Statistical Area 80101100105, as per the above GeoSPARQL example, might be represented as follows:

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix geox: <https://linked.data.gov.au/def/geox#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<https://linked.data.gov.au/dataset/asgs2016/statisticalarealevel1/80101100105> a
geo:Feature ;
  geo:hasGeometry [
    geo:asWKT "<https://w3id.org/dggs/tb16pix>
              POLYGON (P67303515562
                        P67303515565
                        ...
                        P67303523664)"^^geox:wktDGGSLiteral ;
    geox:hasRole <https://linked.data.gov.au/def/geometry-roles/boundary> ;
  ]
.
```

In the above example, the *type* of the DGGs geometry is Polygon and the *role* is Boundary, using a role from the *Geometry Role* vocabulary.

7.7.2. API Features instances

Three instances of the OGC LD API were implemented:

1. **TB16Pix Ref** - TB16Pix Reference Dataset

- Delivers the Zones and Cells of the TB16Pix dataset, i.e. each of its Grids from Grid 0 to Grid 15.
- This dataset contains no data other than the TB16Pix reference grids.

2. **Geofabric** - Australian Hydrological Geospatial Fabric

- A dataset containing a single Collection of hydrological catchments for Australia covering the area of the Australian Capital Territory (ACT) only.
- ~70 features

3. **ASGS** - Australian Statistical Geographies Standard

- A dataset containing a single Collection of Australian census *Statistical Area Level 1* (SA1) blocks covering the area of the Australian Capital Territory (ACT) only.
- ~1,000 features

These are described in [DGGs Enabled Data Services](#).

7.7.3. API functionality

7.7.3.1. Spatial Filtering

In addition to providing information about Datasets, Collections, Feature Lists and Features, the APIs implemented in this Testbed allow for some forms of Feature filtering. This is as required by the OGC API specification and as implemented by many *features* APIs, such as the OGC's Web Feature Service [[WFS](#)].

To filter the Features within a Collections' list of features, query string argument parameters may be used. In regular, non-DGGs, OGC APIs, a WGS84 bounding box filter might look like this:

```
{OGC_API_ENDPOINT}/collections/{COLLECTION_ID}/items?bbox={COORDINATES_OF_A_RECTANGLE}
```

Such filtering is possible within the API instances listed here given that they either store WGS84 data as well as DGGs data or are able to calculate feature geometries expressed in WGS84 coordinates *on the fly*. A working bounding box filter for Features within the ASGS API instance is:

```
http://asgs.surroundaustralia.com/collections/SA1s/items?bbox=149.12037037037038, -35.49268851161001, 148.74999999999997, -35.10748095969024
```

The above URI returns Features (Statistical Area Level 1s) in approximately the northern western third of the Australian Capital Territory.

DGGs filtering is conceptually similar to the above but simpler to communicate and implement. In

the conceptually simplest BBOX implementation, which is operating for the API instances given here, a Cell ID may be given and any Features that have Geometries that overlap with it are returned. The equivalent DGGs Cell ID-based filter to the above bounding box filter is:

```
http://asgs.surroundaustralia.com/collections/SA1s/items?bbox=P673035
```

Here the Cell ID P673035 covers the same area as the WGS84 quadrilateral 149.12037037037038,-35.49268851161001,148.74999999999997,-35.10748095969024 and, indeed, the APIs return the same values for either filter.

Bounding Box filters need not be square: The OGC API - Features indicates that any quadrilateral may be used. Since the rHEALPix DGGs' Cells are square, a more complex filter based on a pair of Cell IDs would need to be implemented to cater for quadrilateral but non-square overlapping areas. Additionally, since DGGs Cells are fixed in space, even if a square bounding box was required that did not align with Cells, such a filter would be required. Quite obviously, such a filter could be of the form:

```
{OGC_API_ENDPOINT}/collections/{COLLECTION_ID}/items?bbox={CELL_ID_1},{CELL_ID_2}
```

Where `CELL_ID_1`,`CELL_ID_2` describe the upper left and bottom right corners of a quadrilateral. Refinement in the bounding box polygon's position can be achieved with higher resolution Cell IDs.

This is not yet implemented in these API instances and is left to future work.

Implementing non-quadrilateral DGGs filters in a manner very similar to non-DGGs polygonal filters is possible. Such filters could be implemented just as traditional polygonal filters are, say a WFS *GetFeature* request using an `ogc:Intersects` payload to communicate the polygonal filter. If characterized in the native DGGs of a queried API a simpler form of communicating the polygon (as opposed to one that needs conversion to the API's native DGGs) could be implemented in a manner similar to the query string approach of `bbox=?=...`. However polygonal description length will be a limitation, as currently with non-DGGs, systems.

7.7.3.2. Non-Spatial Filtering

Many feature APIs allow for filtering by non-spatial attributes of features. For example, the OGC API specification [OAFc] gives the following filter payload example:

```
---
{
  "my_first_parameter": "some value",
  "my_other_parameter": 42
}
---
```

This would be implemented as the following URI query string arguments:
`&my_first_parameter=some%20value&my_other_parameter=42.`

This form of filtering is using context-unaware properties, as per common use of properties in GeoJSON (see the section *Further GeoJSON limitations* above). While this form of filtering could be implemented as readily for DGGs Features as non-DGGs features, the current OGC LD API implementations are predicated on *semantic* data and thus provide for filtering using context-aware properties.

For context-aware filtering of properties, the above example would need to have the property indicators defined. Such parameter definitions are a necessary part of SPARQL ^[14] queries used to query RDF data. An equivalent SPARQL query for the filter in the example above would be:

```
---
PREFIX ex1: <http://example.com/some-ontology_1/>
PREFIX ex2: <http://example.com/some-ontology_2/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
```

```
SELECT ?f WHERE { ?f a geo:Feature ; ex1:my_first_parameter ?param_1 ; ex2:my_other_parameter ?param_2 .
```

```
    FILTER (?param_1 = "some value" && ?param_2 = 42)
}
```

SPARQL or other filtering of features in the APIs within this chapter have not yet been implemented.

This form of parameter defining is bypassed for spatial data due to the presence of the CRS or DGGs dataset identifiers which provide context for the spatial data supplied within the filter.

7.7.4. Supporting Assets

To support these deployments of the OGC API - Features instances accessing DGGs content, a number of semantic assets were generated that describe the data models used within the APIs' data sources. This was required for the APIs also and defined, as supersets of the core OGC API - Features data model. These assets, as well as the API Framework, test client and API instances, are described in [DGGs Enabled Data Services](#).

7.8. OGC API - Processes for DGGs description

As per the Testbed Call for Participation (CFP), the OGC API - Processes implementation in this task supported "the geographic location to zone-ID(s) and reverse conversion", as well as other eventual additions to allow better comparison and discussion.

7.8.1. Basic ideas:

1. The same API elements are used for a DGGs RS service and a DGGs Data service.
2. In the case of a DGGs RS service, the collections are the DGGs instances themselves, and the

returned features have no properties other than the geometry and resolution (e.g., in GeoJSON the properties object will be null).

3. In the case of a DGGs Data service, the collections are the sample DGGs data (hence, made available in a particular DGGs instance, too), and the returned features have actual values associated in the properties object.

7.8.2. Formats:

To cater to different use cases, three different output formats were implemented for each endpoint returning zones:

1. A classic GeoJSON output, with geometry being either the boundary or the center point of the zone (to be controlled with a request parameter), for usage in GIS tools. The longitudes will eventually be extended outside of the -180,180 range to allow visualization by tools that cannot deal properly with dateline crossing.
2. A DGGs JSON output, a small variation of GeoJSON where the geometry is replaced by an array of ZoneIds, was used in both zone-oriented and feature-oriented APIs
3. A plain array of DGGs ZoneIds. This was for cases where the attributes are not needed, and data transfer compactness is paramount.

7.8.3. Questions, discussion topics:

1. Support for multiple DGGs: rHEALPix and H3 were used in order to offer the clients and DGGs data servers at least a DGGs they are not already supporting natively.
2. Is the API going to be sufficiently self-describing such that the clients can figure out everything they need by just walking the API, or will it need specific knowledge contained in an API profile (e.g. a profile for rHEALPix, one for H3, consider for example ZoneId and how to determine them)?
3. For parametric DGGs (e.g. rHEALPix). are specific instances exposed as its own DGGs or somehow are the full parameters of the DGGs in each resource exposed?
4. This API follows a processes API, where most resources are a process and one needs to know valid values to hit them.

7.8.4. Endpoints

Table 5. Endpoint summary

Code	Description
Capabilities	
/	landing page
rootGet	
/conformance	information about specifications that this API conforms to
/collections	the list of supported collections

DGGSAccess	
<code>/collections/{collectionId}</code>	Describes a particular DGGS
<code>/collections/{collectionId}/zones</code>	Access the list of zones in a given DGGS. Can list either all the zones, or a particular subset based on resolution, WGS84 bbox, or list of containing zones (e.g., polygon defined in DGGS terms)
<code>/collections/{collectionId}/zone</code>	Access the definition of a particular zone
<code>/collections/{collectionId}/neighbors</code>	Get the list of neighbouring zones, to a given zone (should it return just a list of identifiers instead of a GeoJSON collection? Could even be a list of links in the zone itself)
<code>/collections/{collectionId}/parents</code>	Get the list of parent zones, to a given zone (should it return just a list of identifiers instead of a GeoJSON collection?)
<code>/collections/{collectionId}/children</code>	Get the list of zones children of a given zone (should it return just a list of identifiers instead of a GeoJSON collection?)
<code>/collections/{collectionId}/point</code>	Returns the id of the zone containing the given point, at the given resolution
<code>/collections/{collectionId}/polygon</code>	Lists zones contained in the polygon

7.8.4.1. Capabilities

7.8.4.1.1. collectionsGet

GET `/collections`

The list of supported collections

Description

Parameters

Return Type

`[collection-list]`

Content Type

- application/json
- text/html

Responses

Table 6. http response codes

Code	Message	Datatype
200	<p>The list of DGGs available and link to the processes. The response contains the list of DGGs objects. This information includes:</p> <ul style="list-style-type: none"> • A local identifier for the DGGs that is unique for this API • An optional human readable title and description for the DGGs • The predominant zone shape (rectangular, triangular, hexagonal) • The list of resolutions for the particular DGGs 	[collection-list]
406	None of the requested media types is supported at the path.	Exception
500	A server error occurred.	Exception

Samples

7.8.4.1.2. conformanceGet

GET /conformance

Information about what standards/specifications this API conforms to.

Description

A list of all conformance classes specified in a standard that the server conforms to.

Parameters

Return Type

ConfClasses

Content Type

- application/json
- text/html

Responses

Table 7. http response codes

Code	Message	Datatype
200	The URIs of all conformance classes supported by the server. To support 'generic'; clients that want to access multiple OGC API - Features implementations - and not 'just'; a specific API / server, the server declares the conformance classes it implements and conforms to.	<i>ConfClasses</i>

406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.1.3. rootGet

GET /

landing page

Description

The landing page provides links to the API definition, the conformance statements and to other resources provided by the API.

Parameters

Return Type

LandingPage

Content Type

- application/json
- text/html

Responses

Table 8. http response codes

Code	Message	Datatype
200	The landing page provides links to the API definition (link relations <i>service-desc</i> and <i>service-doc</i>), the Conformance declaration (path <i>/conformance</i> , link relation <i>conformance</i>), and the Feature Collections (path <i>/collections</i> , link relation <i>data</i>).	<i>LandingPage</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2. DGGSAccess

7.8.4.2.1. collectionsCollectionIdChildrenGet

GET /collections/{collectionId}/children

Get the list of child zones, to a given zone. Question for the DGGs SWG to consider: should it return just a list of identifiers instead of a GeoJSON collection?

Description

Parameters

Table 9. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Table 10. Query Parameters

Name	Description	Required	Default	Pattern
zoneId		X	null	
levels	Number of levels for zone parent/children extration	-	null	

Return Type

ZoneCollectionGeoJSON

Content Type

- application/geo+json
- application/dggs+json
- text/html
- application/json

Responses

Table 11. http response codes

Code	Message	Datatype
200	A list of DGGS zones. The response contains a list of DGGS zones. The response can be a GeoJSON payload with full boundaries, for traditional clients, or a simple list of zone ids, for DGGS aware clients	<i>ZoneCollectionGeoJSON</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2.2. collectionsCollectionIdGet

GET /collections/{collectionId}

Describes a particular DGGS

Description

Parameters

Table 12. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Return Type

Collection

Content Type

- application/json
- text/html

Responses

Table 13. http response codes

Code	Message	Datatype
200	Describes a particular DGGS. This information includes: <ul style="list-style-type: none"> • A local identifier for the DGGS that is unique for this API • An optional human readable title and description for the DGGS • The predominant zone shape (rectangular, triangular, hexagonal) • The list of resolutions for the particular DGGS 	<i>Collection</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2.3. collectionsCollectionIdNeighborsGet

GET /collections/{collectionId}/neighbors

Get the list of neighboring zones, to a given zone. Questions for the DGGS SWG to consider: Should it return just a list of identifiers instead of a GeoJSON collection? Could this even be a list of links in the zone itself?

Description

Parameters

Table 14. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Table 15. Query Parameters

Name	Description	Required	Default	Pattern
zoneId		X	null	
radius	Distance, in zones, from the center zone, to be walked when extracting neighbors. Also known as "k" in a k-ring extraction in some DGGs.	-	null	

Return Type

ZoneCollectionGeoJSON

Content Type

- application/geo+json
- application/dggs+json
- text/html
- application/json

Responses

Table 16. http response codes

Code	Message	Datatype
200	A list of DGGs zones. The response contains a list of DGGs zones. The response can be a GeoJSON payload with full boundaries, for traditional clients, or a simple list of zone ids, for DGGs aware clients.	<i>ZoneCollectionGeoJSON</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2.4. collectionsCollectionIdParentsGet

GET /collections/{collectionId}/parents

Get the list of parent zones, to a given zone. Question for the DGGs SWG to consider: Should it return just a list of identifiers instead of a GeoJSON collection?

Description

Parameters

Table 17. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Table 18. Query Parameters

Name	Description	Required	Default	Pattern
zoneId		X	null	
levels	Number of levels for zone parent/children extraction	-	null	

Return Type

ZoneCollectionGeoJSON

Content Type

- application/geo+json
- application/dggs+json
- text/html
- application/json

Responses

Table 19. http response codes

Code	Message	Datatype
200	A list of DGGs zones. The response contains a list of DGGs zones. The response can be a GeoJSON payload with full boundaries, for traditional clients, or a simple list of zone ids, for DGGs aware clients.	<i>ZoneCollectionGeoJSON</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2.5. collectionsCollectionIdPointGet

GET /collections/{collectionId}/point

Returns the id of the zone containing the given point, at the given resolution

Description

Parameters

Table 20. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Table 21. Query Parameters

Name	Description	Required	Default	Pattern
point	<p>Only zones that contain/touch the point are returned. The point is provided in axis order:</p> <ul style="list-style-type: none"> • Longitude • Latitude <p>The CRS of the values is WGS 84 with axis order longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84). [BigDecimal]</p>	-	null	
resolution		X	null	

Return Type

ZoneCollectionGeoJSON

Content Type

- application/geo+json
- application/dggs+json
- text/html
- application/json

Responses

Table 22. http response codes

Code	Message	Datatype
200	A list of DGGS zones. The response contains a list of DGGS zones. The response can be a GeoJSON payload with full boundaries, for traditional clients, or a simple list of zone ids, for DGGS aware clients	<i>ZoneCollectionGeoJSON</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2.6. collectionsCollectionIdPolygonGet

GET /collections/{collectionId}/polygon

Lists zones contained in the polygon

Description

Parameters

Table 23. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Table 24. Query Parameters

Name	Description	Required	Default	Pattern
bbox	<p>Only features that have a geometry that intersects the bounding box are selected. The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth):</p> <ul style="list-style-type: none"> • Lower left corner, coordinate axis 1 • Lower left corner, coordinate axis 2 • Minimum value, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Maximum value, coordinate axis 3 (optional) <p>The CRS of the values is WGS 84 with axis order longitude/latitude (CRS84) (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different coordinate reference system is specified in the parameter <code>bbox-crs</code>.</p> <p>For WGS 84 longitude/latitude, the values are in most cases the sequence of minimum longitude, minimum latitude, maximum longitude and maximum latitude. However, in cases where the box spans the anti-meridian the first value (west-most box edge) is larger than the third value (east-most box edge).</p> <p>If the vertical axis is included, the third and the sixth number are the bottom and the top of the 3-dimensional bounding box.</p> <p>If a feature has multiple spatial geometry properties, it is the</p>	-	null	

Name	Description	Required	Default	Pattern
polygon	Only zones contained in the polygon are returned. The polygon is specified as WKT. The coordinate reference system of the values is WGS 84 longitude/latitude (CRS84) (http://www.opengis.net/def/crs/OGC/1.3/CRS84).	-	null	
resolution		X	null	

Return Type

ZoneCollectionGeoJSON

Content Type

- application/geo+json
- application/dggs+json
- text/html
- application/json

Responses

Table 25. http response codes

Code	Message	Datatype
200	A list of DGGS zones. The response contains a list of DGGS zones. For traditional clients the response can be a GeoJSON payload with full boundaries, or a simple list of ZoneIds, for DGGS aware clients.	<i>ZoneCollectionGeoJSON</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2.7. collectionsCollectionIdZoneGet

GET /collections/{collectionId}/zone

Access the definition of a particular zone

Description

Parameters

Table 26. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Table 27. Query Parameters

Name	Description	Required	Default	Pattern
zoneId		X	null	

Return Type

ZoneGeoJSON

Content Type

- application/geo+json
- application/dggs+json
- text/html
- application/json

Responses

Table 28. http response codes

Code	Message	Datatype
200	A single DGGS zone The response contains the description of a single DGGS zone.	<i>ZoneGeoJSON</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.4.2.8. collectionsCollectionIdZonesGet

GET /collections/{collectionId}/zones

Access the list of zones in a given DGGS. Can list either all the zones, or a particular subset based on resolution, WGS84 bbox, or list of containing zones (e.g., polygon defined in DGGS terms)

Description

Parameters

Table 29. Path Parameters

Name	Description	Required	Default	Pattern
collectionId	local identifier of a collection	X	null	

Table 30. Query Parameters

Name	Description	Required	Default	Pattern
resolution		X	null	

Name	Description	Required	Default	Pattern
limit	The optional limit parameter limits the number of items that are presented in the response document. Only items on the first level of the collection in the response document are counted. Nested objects contained within the explicitly requested items shall not be counted. Minimum = 1. Maximum = 10000. Default = 10.	-	10	

Return Type

ZoneCollectionGeoJSON

Content Type

- application/geo+json
- application/dggs+json
- text/html
- application/json

Responses

Table 31. http response codes

Code	Message	Datatype
200	A list of DGGS zones. The response contains a list of DGGS zones. For traditional clients the response can be a GeoJSON payload with full boundaries, or a simple list of ZoneIds, for DGGS aware clients	<i>ZoneCollectionGeoJSON</i>
406	None of the requested media types is supported at the path.	<i>Exception</i>
500	A server error occurred.	<i>Exception</i>

Samples

7.8.5. Models

Figure 2 shows the OGC API - Processes class diagram for DGGS.

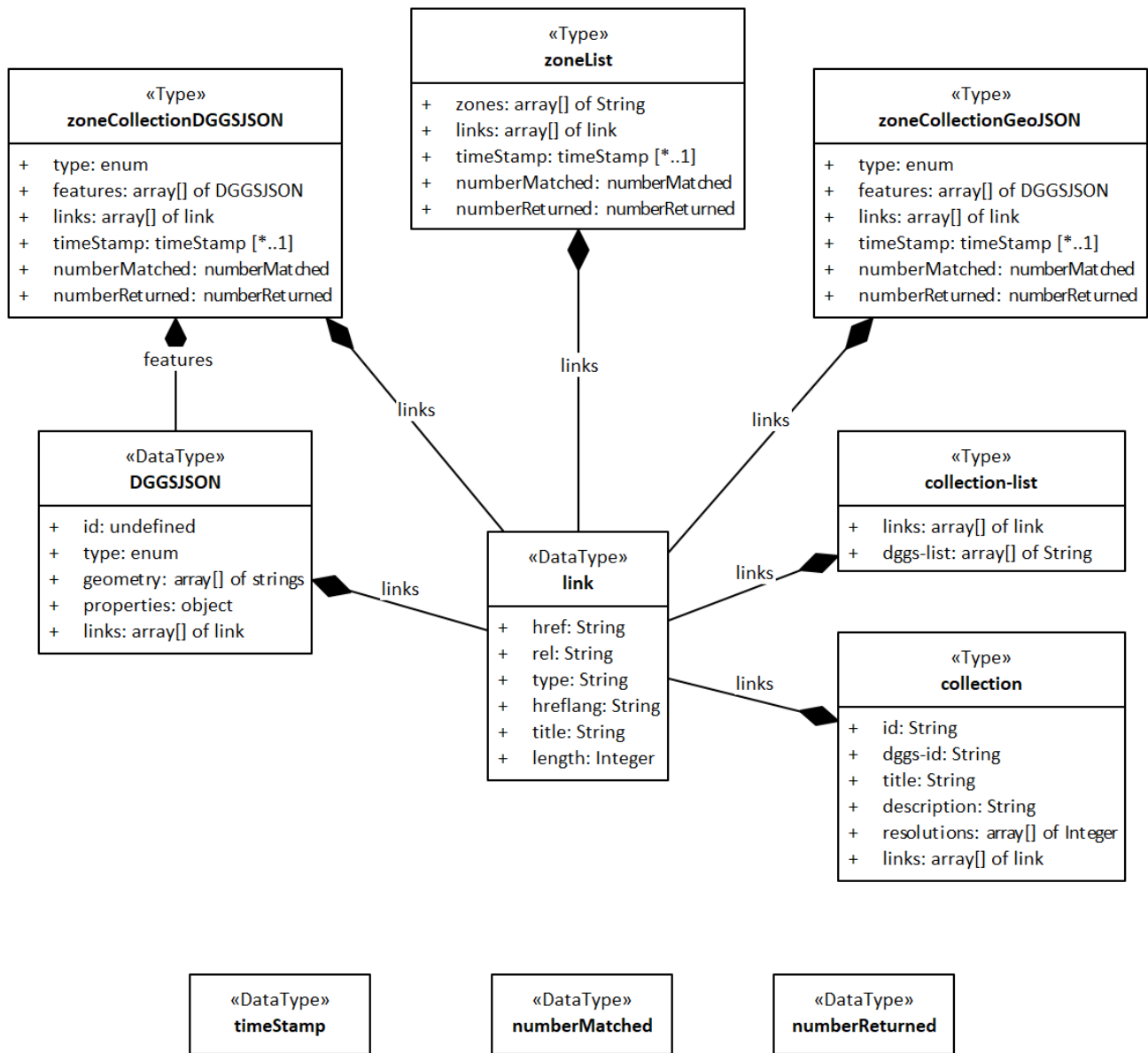


Figure 2. OGC API - Processes class diagram for DGGS.

7.8.5.1. Collection

Field Name	Required	Type	Description	Format
id	X	String	Identifier of the collection. In case the identifier is a DGGS RS service, this will be the DGGS instance identifier, otherwise, the value is going to be the data collection id.	
dggs-id		String	Identifiers of the DGGS instance. For DGGS RS services, this will be the same as the id, for DGGS Data services, dggs-id identifies the particular DGGS instance used.	
title		String	Human readable title of the collection	

Field Name	Required	Type	Description	Format
description		String	A description of the collection	
resolutions		List of [number]		
links	X	List of <i>Link</i>	The list of links, e.g., to the operations provided by this DGGS collection.	

7.8.5.2. *CollectionList*

Field Name	Required	Type	Description	Format
links	X	List of <i>Link</i>		
dggs-list	X	List of [string]		

7.8.5.3. *ConfClasses*

Field Name	Required	Type	Description	Format
conformsTo	X	List of [string]		

7.8.5.4. *DGGSJSON*

Field Name	Required	Type	Description	Format
id	X	oneOf<string,integer>		
type	X	String		<i>Enum:</i> Feature,
geometry	X	List of [string]	The geometry of the feature, as a list of DGGS zone ids	
properties	X	Object		
links		List of <i>Link</i>		

7.8.5.5. *Exception*

Information about the exception: An error code plus an optional description.

Field Name	Required	Type	Description	Format
code	X	String		
description		String		

7.8.5.6. FeatureCollectionGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: Feature Collection,
features	X	List of FeatureGeoJSON		
links		List of Link		
timeStamp		Date	This property indicates the time and date when the response was generated.	date-time
numberMatched		Integer	The number of features of the feature type that match the selection parameters such as bbox .	
numberReturned		Integer	The number of features in the feature collection. If the information about the number of features is not known or difficult to compute a server may omit this information in a response. If the value is provided, the value shall be identical to the number of items in the 'features' array.	

7.8.5.7. FeatureGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: Feature,
geometry	X	geometryGeoJSON		
properties	X	Object		
id		oneOf<string,integer>		
links		List of Link		

7.8.5.8. GeometryGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: GeometryCollection,
coordinates	X	List of [array]		
geometries	X	List of GeometryGeoJSON		

7.8.5.9. GeometrycollectionGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: GeometryCollection,
geometries	X	List of GeometryGeoJSON		

7.8.5.10. LandingPage

Field Name	Required	Type	Description	Format
title		String		
description		String		
links	X	List of Link		

7.8.5.11. LinestringGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: LineString,
coordinates	X	List of [array]		

7.8.5.12. Link

Field Name	Required	Type	Description	Format
href	X	String		

Field Name	Required	Type	Description	Format
rel		String		
type		String		
hreflang		String		
title		String		
length		Integer		

7.8.5.13. MultilinestringGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: MultiLineString,
coordinates	X	List of [array]		

7.8.5.14. MultipointGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: MultiPoint,
coordinates	X	List of [array]		

7.8.5.15. MultipolygonGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: MultiPolygon,
coordinates	X	List of [array]		

7.8.5.16. PointGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: Point,
coordinates	X	List of [number]		

7.8.5.17. PolygonGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: Polygon,
coordinates	X	List of [array]		

7.8.5.18. ZoneCollectionDGGJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: Feature Collection,
features	X	List of DGGJSON		
links		List of Link		
timeStamp		Date	This property indicates the time and date when the response was generated.	date-time
numberMatched		Integer	The number of features of the feature type that match the selection parameters such as bbox .	
numberReturned		Integer	The number of features in the feature collection. If the information about the number of features is not known or difficult to compute a server may omit this information in a response. If the value is provided, the value shall be identical to the number of items in the 'features' array.	

7.8.5.19. ZoneCollectionGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		Enum: Feature Collection,
features	X	List of ZoneGeoJSON		

Field Name	Required	Type	Description	Format
links		List of <i>Link</i>		
timeStamp		Date	This property indicates the time and date when the response was generated.	date-time
numberMatched		Integer	The number of features of the feature type that match the selection parameters such as bbox .	
numberReturned		Integer	The number of features in the feature collection. If the information about the number of features is not known or difficult to compute a server may omit this information in a response. If the value is provided, the value shall be identical to the number of items in the 'features' array.	

7.8.5.20. ZoneGeoJSON

Field Name	Required	Type	Description	Format
type	X	String		<i>Enum:</i> Feature Collection,
features	X	List of <i>FeatureGeoJSON</i>		
links		List of <i>Link</i>		
timeStamp		Date	This property indicates the time and date when the response was generated.	date-time
numberMatched		Integer	The number of features of the feature type that match the selection parameters such as bbox .	
numberReturned		Integer	The number of features in the feature collection. If the information about the number of features is not known or difficult to compute a server may omit this information in a response. If the value is provided, the value shall be identical to the number of items in the 'features' array.	

Field Name	Required	Type	Description	Format
resolution	X	BigDecimal		

7.8.5.21. ZoneGeoJSONAllOf

Field Name	Required	Type	Description	Format
resolution	X	BigDecimal		

7.8.5.22. ZoneList

Field Name	Required	Type	Description	Format
zones	X	List of [string]		
links	X	List of Link		
timeStamp		Date	This property indicates the time and date when the response was generated.	date-time
numberMatched		Integer	The number of features of the feature type that match the selection parameters such as bbox .	
numberReturned		Integer	The number of features in the feature collection. If the information about the number of features is not known or difficult to compute a server may omit this information in a response. If the value is provided, the value shall be identical to the number of items in the 'features' array.	

7.9. Is there a need for an OGC API DGGs implementation standard?

The participant discussions and technology implementations conducted during this Testbed activity evaluated the question of "is there a justification, or need, to draft a separate implementation standard to codify OGC APIs for DGGs?". The work done in both **D137** and **D139** allowed this question to be asked from a number of perspectives.

While the "system" and "data" aspects of a DGGs can be implemented using conventional OGC API - Processes and OGC API - Features schemas, there are some subtle, but very important, differences that separates the operation/access of a DGGs resource from those of a conventional Process or Feature service.

DGGS both simplify the traditional view and add new capability.

The simplification can be summarized as:

1. Very significant reduction in geometric complexity, with a single geometry and topology data model encompassing vector, raster, point cloud, location tags, bounding-box & tiling,
2. Complete separation of geometry from file-type,
3. Extension from 2-D to 3-D & 4D geometries without any additional geometry types or topological queries, and
4. Opportunity to develop a unified spatio-temporal filtering and processing language and associated API for use across all spatial types.

New capabilities include:

1. New unified geometry language to supersede point, line, polygon, raster etc,
2. Explicit alignment of spatio-temporal geometric precision with hierarchical level in the API,
3. Explicit recording and tracking of process history through the API,
4. Leveraging ZoneClass and parent, child hierarchy to implement streaming transmission and recursive API filters and processes, over specified range of levels, and
5. Opportunity to include a desired precision or risk or uncertainty explicitly in spatio-temporal filters and processes.

7.9.1. Proposal for OGC WKT for DGGS geometries

Initial proposal for a new unified geometry language follows. This is intended as a complete list of potential internal DGGS geometry types:

OrdinateList

a list of (1..*) ZoneIds representing a set of points,

- optionally sorted in descending size order,
- optionally compressed to recursively replace complete sets of children by their parents, if compressed a notation to indicate level depth for decompression.
- by default the coordinate will be the representative point nominated by the DGGS, typically the centroid, but some DGGS may also define schema for assigning additional points, such as a cell corner or mid-point on one of the cells' edges. Such schema could for support Arakawa grids and other vertically and temporally staggered grids. Notations for such usage have not been discussed.

DirectedOrdinateList

a list of (2..*) ZoneIds ordered by connectivity representing a sequence of points, ZoneIds in the sequence need not touch, connectivity is assumed to be by 'straight' vectors,

- optionally sorted in groups in descending size order to facilitate streaming transmission and recursive filtering and processing. How to denote a first approximation of a directed sequence of points, and then recursively fill in the detail is not yet resolved. This is analogous

to wavelet approaches.

- optional compression may be possible, but further thinking is needed to elaborate what that means.
- by default the coordinate will be the representative point nominated by the DGGS, typically the centroid. c.f. OrdinateList for possible alternative representative points.

CellList

a list of (1..*) ZoneIds representing a set of nD spaces,

- optionally sorted in descending size order to facilitate streaming transmission and recursive filtering and processing,
- optionally compressed to recursively replace complete sets of children by their parents, if compressed a notation to indicate level depth for decompression.
- optionally sorted in row, column order for an array or image store.

DirectedCellList

a list of (2..*) ZoneIds ordered by connectivity representing a sequence of nD spaces, all ZoneIds in the sequence need to touch at least one other connected ZoneId in the directed sequence, to ensure there are no gaps in the directed sequence,

- optionally sorted in groups in descending size order to facilitate streaming transmission and recursive filtering and processing. c.f. discussion under sorting for DirectedOrdinateLists
- optionally compressed to recursively replace complete sets of children by their parents, if compressed a notation to indicate level depth for decompression.

TileList

a list of (1..*) ZoneIds representing a set of nD bboxes,

- optionally sorted in descending size order,
- optionally compressed to recursively replace complete sets of children by their parents, if compressed a notation to indicate level depth for decompression.
- optionally sorted in row, column order for a tile store.

DirectedTileList

a list of (2..*) ZoneIds ordered by connectivity representing a sequence of nD spaces, ZoneIds in the sequence need not touch, connectivity is assumed to be by 'straight' vectors,

- optionally sorted in groups in descending size order to facilitate streaming transmission and recursive filtering and processing, c.f. discussion under sorting for DirectedOrdinateLists
- optionally compressed to recursively replace complete sets of children by their parents, if compressed a notation to indicate level depth for decompression.

ZoneCollection

a list of (2..*) of one or more of the above types.

- optionally sorted in groups in descending size order to facilitate streaming transmission and

recursive filtering and processing,

- optionally compressed to recursively replace complete sets of children by their parents, if compressed a notation to indicate level depth for decompression. Further work is needed to identify if compression can apply across elements of the collection, or only within elements.

DirectedZoneCollection

a list of one or more of the above types that are connected.

- optionally sorted in groups in descending size order to facilitate streaming transmission and recursive filtering and processing, c.f. discussion under sorting for DirectedOrdinateLists
- optionally compressed to recursively replace complete sets of children by their parents, if compressed a notation to indicate level depth for decompression.

A WKT representation for these geometry type literals needs to be defined for use in GeoSPARQL amongst others.

The draft Topic 21 v2.0 Abstract Specification talks about quantization roles and the following table relates the roles with the resulting data types from the above list. In all instances the quantization role would be expected to be preserved, probably in column definition, so the DGGs knows how to interpret the zone values.

Table 32. WKT geometry object, quantization roles and DGGs geometry

WKT geometry object	DGGs quatization role	DGGs geometry literal	notes and examples
POINT	asCoordinates	OrdinateList	
MULTIPOINT	asCoordinates	OrdinateList	
LINestring	asCoordinates	DirectedOrdinateList	
MULTILINestring	asCoordinates	ZoneCollection of DirectedOrdinateList	
POLYGON	asCoordinates	DirectedOrdinateList	
MULTIPOLYGON	asCoordinates	DirectedZoneCollection of DirectedCoordinateLists	
TRIANGLE	asCoordinates	DirectedOrdinateList	
TIN	asCoordinates	DirectedZoneCollection of DirectedOrdinateList	
POINT	asDataCells	CellList	
MULTIPOINT	asDataCells	CellList	
LINestring	asDataCells	DirectedCellList	
MULTILINestring	asDataCells	ZoneCollection of DirectedCellList	
POLYGON	asDataCells	CellList	

WKT geometry object	DGGS quatization role	DGGS geometry literal	notes and examples
MULTIPOLYGON	asDataCells	CellList	
TRIANGLE	asDataCells	CellList	
TIN	asDataCells	ZoneCollection of CellList	
GEOMETRYCOLLECTION	any	ZoneCollection	

Table 33. Non-WKT geometry objects, quantization roles and DGGS geometry

Geometry object	DGGS quatization role	DGGS geometry literal	notes and examples
point array	asCoordinates	OrdinateList	
point array	asDataCells	CellList	
point cloud	asDataCells	CellList or 3D+T ZoneIds	
raster image	asCoordinates	OrdinateList	
raster image	asDataCells	CellList	
raster image	asDataTiles	TileList	
trajectory	asCoordinates	DirectedOrdinateList of 3D+T ZoneIds	
trajectory	asDataCells	DirectedCellList of 3D+T ZoneIds	
time series	asCoordinates	DirectedOrdinateList of 2D+T or 3D+T ZoneIds	the spatial component may be static
time series	asDataCells	DirectedCellList of 2D+T or 3D+T ZoneIds	the spatial component may be static
WMS image	asGraphicCells	CellList	
WMS image collection	asGraphicCells	ZoneCollection of CellList	
WMS image collection	asGraphicCells	CellList	
WMS image collection	asGraphicTiles	TileList	
Bounding-box	asCoordinates	OrdinateList	
Bounding-box	asDataTile	TileList	
Social media stream	asTags	DataCellList	
Document text	asTags	DataCellList	

Therefore, the Testbed participants recommend that there is significant merit in support of future efforts to develop an OGC API DGGS Standard.

[10] <https://json-ld.org/>

[11] Work in progress on GeoSPARQL 1.1/2.0 is available publicly at <https://github.com/opegeospatial/ogc-geosparql>

[12] https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form

[13] <http://vocabs.gsq.digital/vocabulary/geometry-roles>

[14] <https://www.w3.org/TR/sparql11-query/>

Chapter 8. Use Cases

8.1. Scope of Works under the Testbed-16 CFP

The Testbed 16 Call for Participation (CFP) alluded to non-specific Use Cases that were to demonstrate two possible 'implementation scenarios' involving the following actors:

1. **Server API** - Essentially a reprojection engine that can convert between traditional CRS and DGGs RS and back (Optionally DGGs RS to DGGs RS).
2. **Enabled Service** - Can filter data based on a DGGs zone value (and for the sake of highest testbed value, return these filtered data with zone-IDs).
3. **Client** - Can display DGGs data based on zone-IDs and values.

Based on the CFP, three initial use cases were posed: [Use Case #1 - GPS Location to DGGs Cell...Use Case #3 - Bushfire Impacts from the "Black Summer" Bushfires in Australia](#). Participant discussions matured the use cases and at the same time participants gained confidence in what was possible both from an API perspective. This led, from an implementation perspective for the server and client endpoints, a fourth much more ambitious use case [Use Case #4 - A DGGs version of a DAPA Use Case making use of one or more Jupyter notebooks](#). This last use case is the one the participants finally pursued in the Testbed.

The fourth use case was developed by analyzing the Testbed-16 DAPA thread Use Cases [5] from a DGGs implementation perspective, identifying a set of common elements, and bringing them together as a single Use Case. This Use Case was intended to demonstrate an alternative DGGs roadmap for the Testbed-16 DAPA thread activities.

8.1.1. Use Case #1 - GPS Location to DGGs Cell

1. **Client Receives a GPS Location** - Can be via a direct feed from a GPS Receiver and/or some other data feed containing structured GPS location data.
2. **Client Uses GPS Error Ellipse Information to Query the DGGs Server** - GPS Error Ellipse can be used to determine the appropriate DGGs resolution to query.
3. **DGGs Server Returns the DGGs Cell ID(s) at the Requested Resolution** - DGGs Server returns the DGGs Cell ID (or set of Cell IDs) that best represents the point location plus its horizontal uncertainty.

8.1.2. Use Case #2 - COVID-19 Active Cases Near Me

1. **Official COVID-19 Case Dataset(s) are DGGs Enabled** - Point datasets of COVID-19 cases and their reported location are DGGs Enabled.
2. **Client Includes a Simple Search Facility to Search for Nearby Reported Cases** - Client uses either its' own location (if as a mobile app) or a selected location plus a range distance to search for nearby reported cases of COVID-19.
3. **DGGs Server Returns the DGGs ZoneID(s) for the requested range query** - DGGs Server is queried via a range-type search for DGGs Cell ID's and returns ZoneID's within the search area.

4. **DGGS Enabled Dataset(s) are queried for matching DGGS ZoneID's** - The DGGS Enabled data sources are queried (via conventional Database type of select query) for records "tagged" with the Search Area DGGS ZoneID's.
5. **Client Displays the returned data**

8.1.3. Use Case #3 - Bushfire Impacts from the "Black Summer" Bushfires in Australia

1. **Key Datasets are DGGS Enabled** - Key datasets (e.g. Bushfire affected coverage, road networks, population demographics) are DGGS Enabled.
2. **Client Includes a Simple Search Facility to Search for Nearby Reported Cases** - Client uses a search mechanism (e.g. bbox, range search, etc.) to define a search area for DGGS ZoneID's.
3. **DGGS Server Returns the DGGS ZoneID(s) for the requested range query** - DGGS Server is queried via a range-type search for DGGS ZoneID's and returns ZoneID's within the search area.
4. **DGGS Enabled Dataset(s) are queried for matching DGGS ZoneID's** - The DGGS Enabled data sources are queried (via conventional Database type of select query) for records "tagged" with the Search Area DGGS ZoneID's.
5. **Client Displays the returned data**

8.1.4. Use Case #4 - A DGGS version of a DAPA Use Case making use of one or more Jupyter notebooks

1. **Key Datasets are DGGS Enabled** - Selected EO DAPA data sources are converted to DGGS form, potentially using Jupyter notebook to 'read' the DAPA data.
 - a. Question arises as to which derived DAPA EO data is available globally rather than only in the specific region of the DAPA use case.
 - b. Fundamental difference between the DAPA pre-processing and pre-processing for DGGS is that the DGGS quantization process will populate a DGGS resolution that is similar to the EO resolution, and then recursively aggregate it to n coarser resolutions with a suite of summary statistics, to be stored in the DGGS Process Server,
 - i. Australian statistical mesh-block polygons are converted to DGGS form, to be stored in the DGGS Feature Server,
 - ii. Australian catchments are converted to DGGS form, to be stored in the DGGS Feature Server,
2. **Client Performs a Feature Query using a polygon selection from the Australian mesh-blocks or catchments as AOI** - Client uses a DGGS Feature search mechanism (e.g. bbox, range search, etc...) to define a AOI which is returned as DGGS ZoneID's. Stretch goal, AOI filter includes time.
3. **Client displays the AOI and underlying DGGS polygon data on a map display** - Part of the Jupyter notebook.
4. **Client Sends a Process Query using the returned DGGS ZoneID list to construct a low resolution AOI query** - Using the Jupyter notebook the user selects the process to be performed and the desired resolution, and the Client issues the Process Request to the DGGS Process

Server.

5. **DGGS Process Server returns EO summary statistics for the chosen AOI at the chosen resolution** - DGGS Process Server computes the chosen process over the query AOI and returns the requested processing.
6. **Client Displays the returned data**
7. **Repeat steps (4) to (6) as required until they are satisfied with the chosen process, and have achieved the desired precision (resolution)**

NOTE

There is a potential for precision hints, and incorporating DAPA style EO analysis, because GeoServer already has the processing engine.

Major caveat

Knowing when to stop work in this Testbed thread and what to hand over to the next DGGS Testbed activity. This is because there is a very clear understanding that there are many aspects to compare and contrast and explore. Also remembering that at the start of the TB-16 DGGS thread, the idea of 'DGGS data' had not been fully anticipated, and so there was no sponsor expectation of having 'DGGS data'. At this point, the use and creation of DGGS Data is fundamental to the TB-16 DGGS thread and both the APIs.

Chapter 9. DGGs Server and API

GeoSolutions delivered the **D137 DGGs Server Implementation**: Open-source server implementation with support for DGGs API.

In particular, the delivery was comprised of four GeoServer community modules and made available:

- As source code, licensed under the GPL license, in the GeoServer GIT repository, [under the umbrella of OGC API modules](https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/dggs) [https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/dggs].
- As binary extensions, ready to be dropped into a GeoServer installation, as part of the [larger OGC API family test packages](https://build.geoserver.org/geoserver/master/community-latest/geoserver-2.19-SNAPSHOT-ogcapi-plugin.zip) [https://build.geoserver.org/geoserver/master/community-latest/geoserver-2.19-SNAPSHOT-ogcapi-plugin.zip]. The zip files enable, among others, support for OGC API - Features, Tiles, Styles and DGGs.
- As a live server for tests, covering all for live tests on the [GeoSolutions Testbed-16 activities](https://tb16.geo-solutions.it/geoserver/web/) [https://tb16.geo-solutions.it/geoserver/web/].

In particular, the Java modules are:

- **dggs-core**: Provides the basic Java API for DGGs, implementations of said interfaces for H3 and rHEALPix. Also, a basic GeoTools **DataStore** implementation exposing the DGGs zones as collections of OGC features, for publication on all classic OGC protocols, as well as the DGGs API was provided.
- **ogcapi-dggs**: Exposes the contents of suitably set-up [ClickHouse database](https://clickhouse.tech/) [https://clickhouse.tech/] as OGC features, for publication on all classic OGC protocols, as well as the DGGs API.
- **ogcapi-dggs**: Implementation of the DGGs API exposing all available DGGs enabled layers.
- **web-dggs**: Provides the user interface for configuring the above **DataStore** implementations in the GeoServer administration console.

The following sections provide details on every component.

9.1. DGGs library comparisons and choices

The GeoTools library and GeoServer web application had no support for DGGs at the beginning of the testbed. Three DGGs libraries have been evaluated in order to establish a baseline for a generic DGGs Java API.

In particular the following libraries were evaluated:

- **H3** [https://h3geo.org/docs]: Provides support for the same named DGGs, is a library implemented entirely in C, but providing bindings for a variety of other languages, including, among others, **Java** [https://github.com/uber/h3-java], **Python** [https://github.com/uber/h3-py], **JavaScript** [https://github.com/uber/h3-js] and **Go** [https://github.com/uber/h3-go].
- **rhealpixdggs-py** [https://github.com/manaakiwhenua/rhealpixdggs-py]: The reference implementation

of rHEALPix in Python.

- **open-eaggr** [<https://github.com/riskaware-ltd/open-eaggr>]: Provides support for the OpenEAGGR DGGS, is a library implemented in C, but also providing bindings for Java and Python.

Table 34 compares how the three libraries represent the basic DGGS concepts and the integration with geographic coordinates and basic geometries is interesting.

Table 34. Basic DGGS concepts in different libraries

Class/Concept	H3	OpenEAGGR	rHEALPix
Main Class	H3Core [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java]	Eaggr [https://github.com/riskaware-ltd/open-eaggr/blob/v2.0/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java]	RHEALPixDGGS [https://github.com/manaakiwhenua/rhealpixdggs-py/blob/0.5.3/rhealpixdggs/dggs.py#L197]
DGGS Zone	A long or hex string (no dedicated object, all functionality is provided via H3Core)	DggsCell [https://github.com/riskaware-ltd/open-eaggr/blob/v2.0/EAGGRJava/src/uk/co/riskaware/eaggr/DggsCell.java] class (wraps the zone identifier, a string)	Cell [https://github.com/manaakiwhenua/rhealpixdggs-py/blob/0.5.3/rhealpixdggs/dggs.py#L1525] class. Fields include zone identifier id , source DGGS, ellipsoid , resolution .
Geographic point	GeoCoord [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/util/GeoCoord.java] class, as lat and lon (no resolution)	LatLongPoint [https://github.com/riskaware-ltd/open-eaggr/blob/v2.0/EAGGRJava/src/uk/co/riskaware/eaggr/LatLongPoint.java] as lat , lon and resolution	Python tuple with lat and lon (no resolution)
Geographic line	Array of GeoCoord	LatLongLineString [https://github.com/riskaware-ltd/open-eaggr/blob/v2.0/EAGGRJava/src/uk/co/riskaware/eaggr/LatLongLinestring.java], containing a list of LatLongPoint	Array of Python tuples, each representing a single point.
Geographic polygon	Array of arrays of GeoCoord (to represent shell and holes)	LatLongPolygon [https://github.com/riskaware-ltd/open-eaggr/blob/v2.0/EAGGRJava/src/uk/co/riskaware/eaggr/LatLongLinestring.java], able to represent a polygon with holes.	Array of point tuples (no support for polygons with holes)

Table 35 compares the libraries in terms of functionality provided. The following table provides a set of links to tagged versions of the respective software, with eventual comments when the method in question has unusual behavior.

Table 35. Basic DGGS functions in different libraries

Function	H3	OpenEAGGR	rHEALPix
Lat/Longitude/Resolution to zone identifier	geoToH3(lat,lon,res): long [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L158]	convertPointToDggsCell (LatLongPoint): DggsCell [https://github.com/riskaware-ltd/open-eaggr/blob/master/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java#L108]	cell_from_point(resolution, point): Cell [https://github.com/manaakiwhenua/rhealpixdggs-py/blob/0.5.3/rhealpixdggs/dggs.py#L773]
Zone to its center (geographic point)	h3ToGeo(long): GeoCoord [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L191]	convertDggsCellToPoint (DggsCell cell): LatLonPoint [https://github.com/riskaware-ltd/open-eaggr/blob/master/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java#L248]	Cell.centroid [https://github.com/manaakiwhenua/rhealpixdggs-py/blob/0.5.3/rhealpixdggs/dggs.py#L2567]
Zone to polygon boundary	h3ToGeoBoundary(long): GeoCoord [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L211]	convertDggsCellOutlineToShapeString(DggsCell cell, ShapeStringFormat format): String [https://github.com/riskaware-ltd/open-eaggr/blob/master/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java#L388]. Returns either KML, GeoJSON, or WKT.	Cell.boundary(n, plane, interior) [https://github.com/manaakiwhenua/rhealpixdggs-py/blob/0.5.3/rhealpixdggs/dggs.py#L2275], or RHEALPixDGGS.vertices() [https://github.com/manaakiwhenua/rhealpixdggs-py/blob/0.5.3/rhealpixdggs/dggs.py#L2166]. Returns an array of tuples representing points.
Zone resolution	h3GetResolutions(id) [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L760]	No direct API, convertDggsCellToPoint returns a LatLonPoint , which in turn contains a resolution.	Cell.resolution [https://github.com/manaakiwhenua/rhealpixdggs-py/blob/0.5.3/rhealpixdggs/dggs.py#L1541]

Function	H3	OpenEAGGR	rHEALPix
Zone to parent	h3ToParent(long, parentRes): long [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L781], returns the parent at the given resolution.	getCellParents:DggsCell [https://github.com/riskaware-ltd/open-eaggr/blob/master/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java#L444]] returns the direct parents of the zone (in this DGGs, they may be more than one)	No direct API, just eliminate the last char in the zone identifier.
Zone to children	h3ToChildren(long, childResolution): long[] [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L821]] returns all the children at the given target resolution.	getCellChildren:DggsCell [https://github.com/riskaware-ltd/open-eaggr/blob/master/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java#L486]] returns the direct children of a cell	Cell.subcells(resolution) [https://github.com/manaakiwhenua/rhealpixdgg-py/blob/0.5.3/rhealpixdgg/dggs.py#L1958] returns all the children at the given target resolution
Neighboring zone	kRing(long, numRings) → long[] [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L261]] Returns all the zones within K zones from the given center, organized by rings.	getCellSiblings(DggsCell) → DggsCell[] [https://github.com/riskaware-ltd/open-eaggr/blob/master/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java#L517]]	Cell.neighbors [https://github.com/manaakiwhenua/rhealpixdgg-py/blob/0.5.3/rhealpixdgg/dggs.py#L2722] full direct neighbors of the zone (zones sharing a side with the center one)
Polygon to set of zones	polyfill(GeoCoord[] shell, GeoCoord[][] holes, res) , all the cells contained in the given polygon shell, at the given resolution.	convertShapesToDggsShapes(LatLonShape[] DggsShape[]) [https://github.com/riskaware-ltd/open-eaggr/blob/master/EAGGRJava/src/uk/co/riskaware/eaggr/Eaggr.java#L142]]): DggsShape[]	minimal_cover(resolution, points) [https://github.com/manaakiwhenua/rhealpixdgg-py/blob/0.5.3/rhealpixdgg/dggs.py#L1313]]

Function	H3	OpenEAGGR	rHEALPix
Compact (given a set of zone identifiers, return the minimal set of zones representing the same area, using zones at lower resolution)	compact(long[] [https://github.com/uber/h3-java/blob/v3.6.4/src/main/java/com/uber/h3core/H3Core.java#L902]) : long[]	Not supported	Not supported

Finally, given the target of integration with GeoServer, evaluating the suitability of integration with a Java Web Server was important.

- H3 provides the most natural integration with Java among the libraries explored. The Java bindings contain, in the 700kb library JAR, native binary implementations for Windows, Linux, OSX and Android on a variety of CPU architectures. On most platforms using the library is simply a matter of setting the library as a dependency, and using the library as if it was a pure Java one. No external setups or native builds were required. A session of load testing proved the library to be very stable. No crashes were detected.
- OpenEAGGR provides a Java binding for the C library. However, the latter has to be custom built for the target platform, from the sources. Load tests on a Linux Ubuntu 64bit derivative showed a tendency to throw segmentation faults, which as a consequence crashed the entire virtual machine.
- rHEALPix is a Python 3 library, with [scipy](https://www.scipy.org/) [https://www.scipy.org/] dependencies and integration with a small C library implementing the rHEALPix projection calculations. As a result, this library can be used solely in a native Python environment. This is also due to the fact that as alternatives for the Java Virtual Machine (JVM) such as [Jython](https://www.jython.org/) [https://www.jython.org/] cannot be used. The [JEP library](https://github.com/ninia/jep) [https://github.com/ninia/jep] was chosen for the integration, as it provides the ability to call onto a native Python interpreter from Java code.

As a result of the above tests, and considering the time limits of a Testbed, the OpenEaggr library was dropped from the actual DGGS integration experiments.

The rHEALPix integration development produced functional code, with a couple of significant limitations:

- The integration proved to be slow. In particular, computing the boundary and center of rHEALPix zones is too slow for significant production usage.
- The integration did not scale up with concurrent requests due to an incompatibility between the Java web server runtime operations (based on threads) and the Python [Global Interpreter Lock \(GIL\)](#) [<https://github.com/ninia/jep/wiki/Jep-and-the-GIL>], disallowing concurrent operations among Python interpreters running in the same process.

The H3 integration has showed no issues so far, being stable, fast and linearly scalable.

9.2. DGGS Java API

The DGGS subsystem in GeoServer is based on a pluggable extension point called [DGGSInstance](#) [<https://github.com/geoserver/geoserver/blob/06b043933b9f89933af40fa4e8573252b00a4b98/src/community/ogcap/dggs/dggs-core/src/main/java/org/geotools/dggs/DGGSInstance.java>]. The interface exposes the basic operations of a DGGS, allowing upstream code to operate against a DGGS without relying on its implementation details:

- List of resolution levels, as integer numbers.
- Getting a zone from a string identifier.
- Getting all the zones intersecting a given envelope, at a given target resolution, with eventual compaction.
- Getting the neighbors of a zone, with a given radius, expressed as number of zones
- Getting the children of a zone, at a given target resolution.
- Getting all the parents of a given zone, recursively (the number of resolution is usually small, implying a small number of potential parents as well).
- Mapping a point to the zone containing it, at a given resolution.
- Mapping a polygon to a list of zones at a given resolution, with eventual compaction.

The [DGGSInstance](#) implementations can be discovered at runtime using the Java [Service Provider Interfaces \(SPI\)](#) [<https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html>] mechanism. This allows creating new DGGS integrations, packaging them as JAR files, and having GeoServer discover their presence.

9.3. The DGGS geometry store

In GeoTools and GeoServer a [DataStore](#) [<https://docs.geotools.org/latest/userguide/library/data/datastore.html>] is a Java interface " used to access and store geospatial data in a range of vector formats".

A [DataStore](#) provides access to a set of feature collections, enabling exploration of their structure

(list of attributes and type), as well as to query them, and optionally, to modify their contents. The operation model of the **DataStore** is heavily influenced by the original WFS 1.0 design, and is still a good match for the existing OGC API - Features.

In GeoServer **DataStore** instances can be plugged in, discovered and configured at runtime using the SPI mechanism. This allows publishing data over various OGC protocols, such as WFS, WMS, WMTS and the OGC API equivalents.

The first DGGs integration with GeoServer was thus the **DGGsGeometryStore** [<https://github.com/geoserver/geoserver/tree/06b043933b9f89933af40fa4e8573252b00a4b98/src/community/ogcapi/dggs/dggs-core/src/main/java/org/geotools/dggs/gstore>], exposing the DGGs zones as features, with an identifier (the ZoneId), a resolution, and a geometry (the boundary).

The store currently takes as a parameter the DGGs to be used. In the future configuration parameters for DGGs like rHEALPix will be exposed as well. The current rHEALPix implementation exposes the configuration for the TB16Pix RS.

Edit Vector Data Source

Edit an existing vector data source

DGGs Geometry Store

Store returning DGGs zones, with no data associated (pure geometric description)

Basic Store Info

Workspace *

dggs

Data Source Name *

H3

Description

Enabled

Connection Parameters

DGGs factory identifier *

H3

rHEALPix

H3

Apply

Cancel

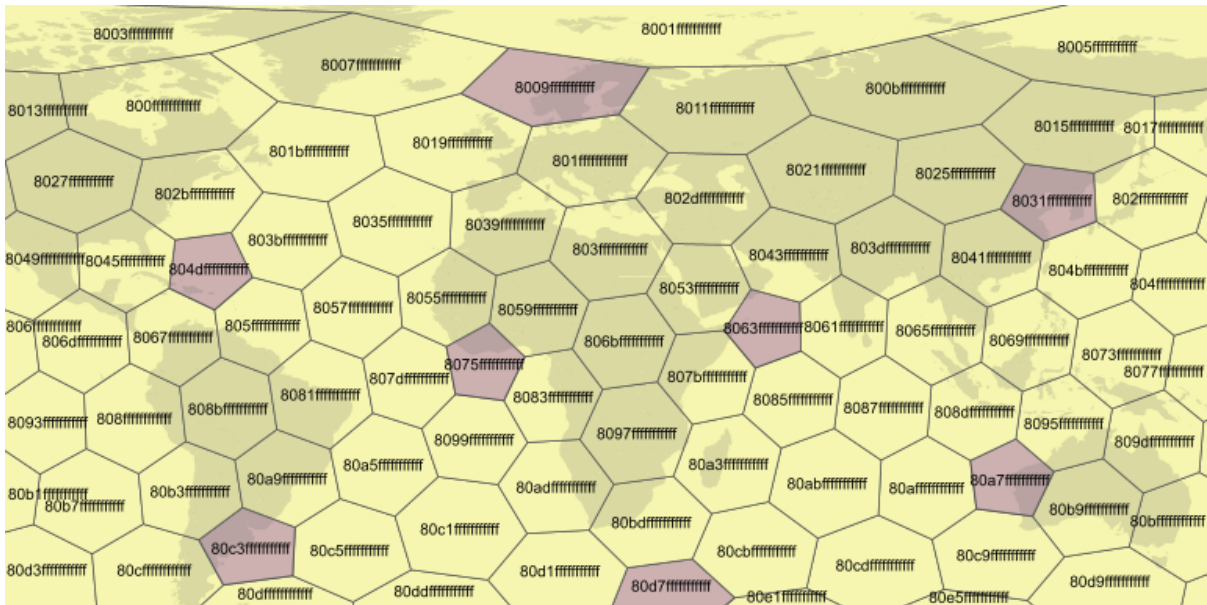
Figure 3. Setting up the DGGs geometry store, with a choice of DGGs to use

The store then exposes a single feature collection, with the same name as the DGGs, that can be configured as a layer in GeoServer, and then consumed from services such as WMS and WFS.

An example WMS request could be:

https://tb16.geo-solutions.it/geoserver/dggs/wms?STYLES=&LAYERS=dggs%3ATB16-Pix%2Cdggs%3Aworld&EXCEPTIONS=application%2Fvnd.ogc.se_inimage&FORMAT=image%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&SRS=EPSG%3A4326&BBOX=-106.875,-45.3515625,163.125,89.6484375&WIDTH=768&HEIGHT=384

The following images show maps of both H3 and TB16-Pix from the WMS server:



```

{
  "type": "FeatureCollection",
  "features": [{
    "type": "Feature",
    "id": "TB16-Pix.S",
    "geometry": {
      "type": "Polygon",
      "coordinates": [
        [
          [-180, -90],
          [-180, -41.9379],
          [180, -41.9379],
          [180, -90],
          [-180, -90]
        ]
      ]
    },
    "geometry_name": "geometry",
    "properties": {
      "zoneId": "S",
      "resolution": 0,
      "shape": "cap",
      "color": "#FF80FF"
    },
    "bbox": [-180, -90, 180, -41.9379]
  }],
  "totalFeatures": 6,
  "numberMatched": 6,
  "numberReturned": 1,
  "timeStamp": "2020-10-14T13:53:11.554Z",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::4326"
    }
  },
  "bbox": [-180, -90, 180, 90]
}

```

When using a WMS endpoint, the geometry store dynamically chooses which resolution level to return based on the current scale denominator.

The resolution level can be controlled via the `viewparams` vendor parameter. This allows passing key values to the data sources. Originally conceived to expand variables in layers sourced from [SQL statements](https://docs.geoserver.org/stable/en/user/data/database/sqlview.html) [https://docs.geoserver.org/stable/en/user/data/database/sqlview.html], the `viewparams` mechanism is now available to every store.

In particular, the `resOffset` view parameter can be used to offset the target resolution. This can be used, for example, to display the DGGS layer multiple times, and viewing the parent/child

relationship in a WMS display:

- [H3 live link](https://tb16.geo-solutions.it/geoserver/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&FORMAT=application/openlayers&TRANSPARENT=true&VIEWPARAMS=resOffset%3A0%2CresOffset%3A-1%2CresOffset%3A-2%2C%2C&STYLES=%2Cdggs_m1%2Cdggs_m2%2C%2C&LAYERS=dggs%3AH3%2Cdggs%3AH3%2Cdggs%3AH3%2Cworld,dggs%3Aact&exceptions=application%2Fvnd.ogc.se_inimage&SRS=EPSG%3A4326&WIDTH=1200&HEIGHT=700&BBOX=-180,-90,180,90) [https://tb16.geo-solutions.it/geoserver/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&FORMAT=application/openlayers&TRANSPARENT=true&VIEWPARAMS=resOffset%3A0%2CresOffset%3A-1%2CresOffset%3A-2%2C%2C&STYLES=%2Cdggs_m1%2Cdggs_m2%2C%2C&LAYERS=dggs%3AH3%2Cdggs%3AH3%2Cdggs%3AH3%2Cworld,dggs%3Aact&exceptions=application%2Fvnd.ogc.se_inimage&SRS=EPSG%3A4326&WIDTH=1200&HEIGHT=700&BBOX=-180,-90,180,90] displaying zones, their parent and grand-parent (zoom in a few times in order for the three levels to display)
- [TB16-Pix live link](https://tb16.geo-solutions.it/geoserver/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&FORMAT=application/openlayers&TRANSPARENT=true&VIEWPARAMS=resOffset%3A0%2CresOffset%3A-1%2CresOffset%3A-2%2C%2C&STYLES=%2Cdggs_m1%2Cdggs_m2%2C%2C&LAYERS=dggs%3ATB16-Pix%2Cdggs%3ATB16-Pix%2Cdggs%3ATB16-Pix%2Cworld,dggs%3Aact&exceptions=application%2Fvnd.ogc.se_inimage&SRS=EPSG%3A4326&WIDTH=1299&HEIGHT=700&BBOX=-180,-90,180,90) [https://tb16.geo-solutions.it/geoserver/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&FORMAT=application/openlayers&TRANSPARENT=true&VIEWPARAMS=resOffset%3A0%2CresOffset%3A-1%2CresOffset%3A-2%2C%2C&STYLES=%2Cdggs_m1%2Cdggs_m2%2C%2C&LAYERS=dggs%3ATB16-Pix%2Cdggs%3ATB16-Pix%2Cdggs%3ATB16-Pix%2Cworld,dggs%3Aact&exceptions=application%2Fvnd.ogc.se_inimage&SRS=EPSG%3A4326&WIDTH=1299&HEIGHT=700&BBOX=-180,-90,180,90] displaying zones, their parent and grand-parent.

A sample URL is URL-decoded and dissected below:

URL component	Description
https://tb16.geo-solutions.it/geoserver/wms?	Host and path
SERVICE=WMS	
VERSION=1.1.1	
REQUEST=GetMap	
TRANSPARENT=true	
BBOX=-180,-90,180,90	Whole world map
SRS=EPSG:4326	In plate carree.
WIDTH=1299	
HEIGHT=700	
exceptions=application/vnd.ogc.se_inimage	
FORMAT=application/openlayers	Returns a working OpenLayers client displaying the same map
LAYERS=dggs:TB16-Pix,dggs:TB16-Pix,dggs:TB16-Pix,world	Superimposes three times the TB16-Pix layer, overlays a transparent world continents layer
STYLES=,dggs_m1,dggs_m2,,	Changes the style of the second and third TB16-Pix layers, displaying the direct parent border as thick orange, and the grandparent as thick red.
VIEWPARAMS=resOffset:0,resOffset:-1,resOffset:-2,	The first TB16-Pix layer uses the "natural" resolution, as decided by the <code>DGGSGeometryStore</code> , the second shows one resolution level less, and then third, two resolution levels less

Static displays of the parent/child geometric relationships in H3 and TB16-Pix follow.

BackgroundOtherIndices.html#Normaliz3], computed as $(B11 - B08) / (B11 + B08)$

- [The Normalized Difference Water Index](https://en.wikipedia.org/wiki/Normalized_difference_water_index) [https://en.wikipedia.org/wiki/Normalized_difference_water_index], computed as $(B03 - B08) / (B03 + B08)$

To ensure enough data coverage, a bounding box slightly larger than the ACT has been used, in order to ensure enough coverage for the higher level parent zones. Figure 8 shows a reference display of the area.

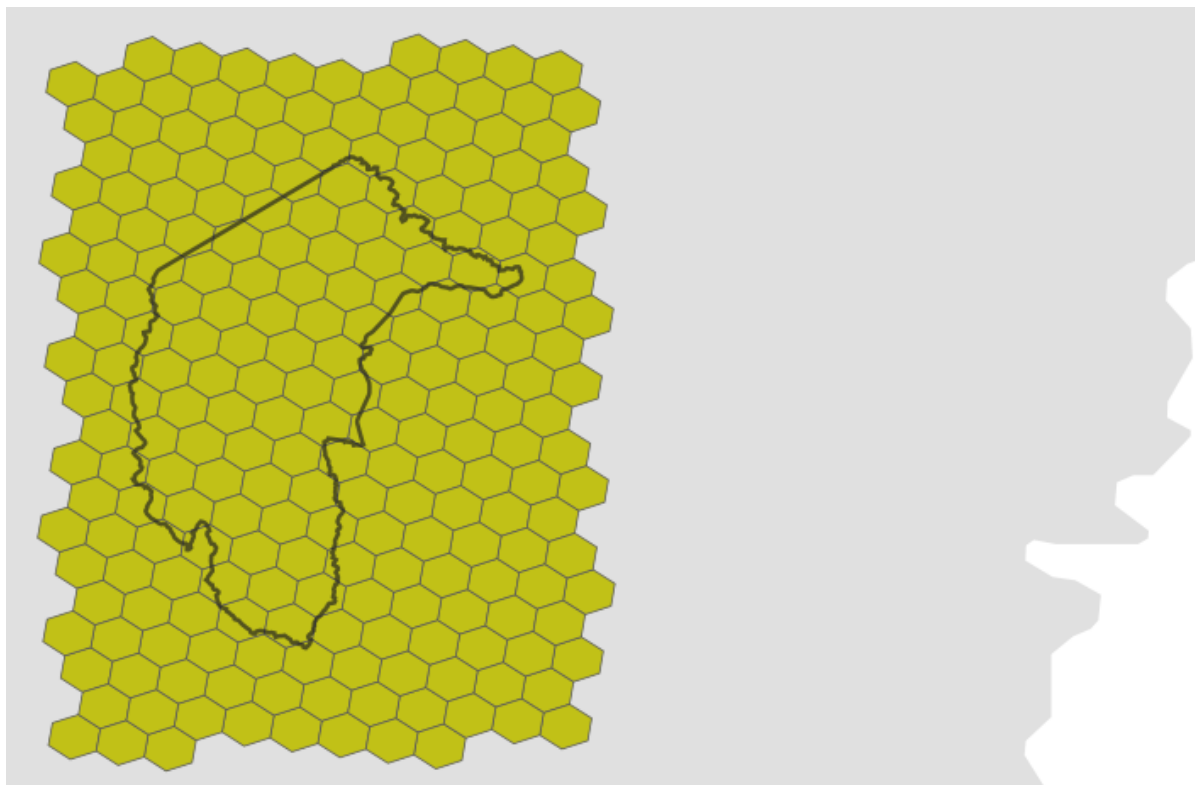


Figure 8. Displaying the H3 grid over ACT

Even with such a small area, and reduced set of resolutions, the number of records to be stored is significant. In addition to that, to exercise the DGGS and DAPA API at multiple times, snapshots of the area have been taken in 2019 and 2020.

Table 36 & Table 37 report the number of actual zones imported per level (data import is discussed later):

Table 36. H3 zone count over sampling area (two time slices)

Resolution	Zone count
5	44
6	450
7	3.494
8	25.462
9	180.772
10	1.272.272

Resolution	Zone count
11	8.923.852

Table 37. TB16Pix zone count over sampling area (two time slices)

Resolution	Zone count
5	8
6	112
7	1.188
8	11.152
9	104.000
10	940.252
11	8.470.532

This clarifies the requirements on the data storage, in particular:

- Ability to efficiently store in excess of 20 million records, each having 16 numerical attributes.
- Compact storage.
- Ability to efficiently locate records by zone identifiers, or by n-th parent zone id.
- Ability to quickly perform aggregations over columns values, such as extracting the maximum, minimum, average and sum of values in a given polygon (expressed as sets of zoneIds to retrieve).

A classic relational database can still handle 20 million records at ease. However, with larger areas or deeper resolution sampling, the number would grow very large. The whole world sampled at the highest resolution level of H3 would require storing 597 trillion records.

For the use case, a dedicated On-Line Analytical Processing (OLAP) database is probably a better choice. [ClickHouse](https://clickhouse.tech/) [https://clickhouse.tech/] was the choice to demonstrate this option. The [following characteristics](https://clickhouse.tech/docs/en/introduction/distinctive-features/) [https://clickhouse.tech/docs/en/introduction/distinctive-features/] make ClickHouse a good match for the use case:

- Tables are natively partitioned in sub-sets by a given partitioning key, supporting splitting a large dataset in smaller, more manageable parts.
- Compressed column oriented storage, saving data by column instead of by row, indexing it by the given table key, and compressing the data for efficient storage.
- Partitioning can be extended to multiple nodes.
- Designed as an OLAP database, can perform aggregation quickly, distributing the calculation over the different nodes, and for each node, using all available CPUs

Unlike other recent OLAP database, such as [QuestDB](https://questdb.io/) [https://questdb.io/], the partitioning key is free

to use any column designed, thus supporting splitting the data over both space and time. The following is an example of table creation, using the simplest partitioning engine available in ClickHouse:

```
CREATE TABLE IF NOT EXISTS s2
(
  `zoneId` String,
  `resolution` UInt8,
  `date` DateTime,
  `B01` UInt16,
  `B02` UInt16,
  `B03` UInt16,
  `B04` UInt16,
  `B05` UInt16,
  `B06` UInt16,
  `B07` UInt16,
  `B08` UInt16,
  `B09` UInt16,
  `B10` UInt16,
  `B11` UInt16,
  `B12` UInt16,
  `B8A` UInt16,
  `NDVI` Float64,
  `NDBI` Float64,
  `NDWI` Float64
)
ENGINE = MergeTree()
PARTITION BY (substring(zoneId, 1, 6), date)
ORDER BY (resolution, date, zoneId)
```

Highlights:

- The partitioning engine is mandatory in ClickHouse, all tables have to be partitioned.
- The partitions are constructed using the first six characters of the zoneId, and the date.
- Each partition's contents are sorted by resolution, date and zoneId.

This setup allows to quickly resolve queries like the following, extracting all the children of zone **P57624** at resolution 10.

```
SELECT *
FROM s2
WHERE zoneId like 'P57624%'
AND resolution = 10
```

For reference, the above structure supported storing all zones in the test area — around 10 million per DGGS type — in roughly 500MB per table. This is a couple of times larger than an equivalent compressed raster storage covering the same area ^[15], at the same resolution. However, there is

advantage of a fast SQL aggregation engine built-in, and the freedom to mix columns of different data types.

The following query, computing the maximum value of the **B01** over the 8 million zones covering the test area at zoom level 11, ran in 0.015 seconds ^[16]:

```
SELECT avg(B01)
FROM act_h3.s2
WHERE resolution = 11
```

9.5. Importing data in ClickHouse

As part of the Testbed activities, GeoSolutions imported Sentinel 2 data covering the test area over two time slices, once for H3, and once for TB16-Pix.

The data import was performed using the following steps:

- Using GDAL to translate the Sentinel 2 JPEG 2000, split band rasters into 13 bands TIFFs
- Using Java and GeoTools for mosaicking and reading the Sentinel 2 data, sampling it at the center of each resolution 11 zone intersecting the test area, computing the NDVI/NDBI/NDWI indexes, and inserting the result in ClickHouse
- Performing simple aggregation queries to compute the average value of each band and index in parent cells, based on the values of the children cells.

In particular, the Java program for H3 used 16 parallel threads to perform the sampling, loading the 9 million zones in the database in 25 seconds. The generation of all upper levels, by aggregation query in ClickHouse, completed in 1.5 seconds.

The following sample query created all resolution 10 H3 zones, aggregating the values of their direct children at resolution 11, but only when all the children of a given parent zone were available. The query leverages ClickHouse native H3 support functions:

```

INSERT INTO s2
  SELECT h3ToString(h3ToParent(stringToH3(zoneId), toUInt8(resolution - 1))),
    resolution - 1,
    max(date),
    round(avg(B01)),
    round(avg(B02)),
    round(avg(B03)),
    round(avg(B04)),
    round(avg(B05)),
    round(avg(B06)),
    round(avg(B07)),
    round(avg(B08)),
    round(avg(B09)),
    round(avg(B10)),
    round(avg(B11)),
    round(avg(B12)),
    round(avg(B8A)),
    (avg(B08) - avg(B04)) / (avg(B08) + avg(B04)),
    (avg(B11) - avg(B08)) / (avg(B11) + avg(B08)),
    (avg(B03) - avg(B08)) / (avg(B03) + avg(B08))
  FROM s2
  where resolution = 11
  and date = '2020-09-06 00:00:00'
  group by h3ToParent(stringToH3(zoneId), toUInt8(resolution - 1)), resolution
  having count(*) = length(h3ToChildren(h3ToParent(stringToH3(zoneId), toUInt8
(resolution - 1)), resolution))

```

The equivalent rHEALPix query can be built using simple string manipulations (the parent of a zone can be obtained by removing the last character from the zone id) and has a simpler "having" condition, considering a rHEALPix zone always has 9 children:

```

INSERT INTO s2
  SELECT substring(zoneId, 1, length(zoneId) - 1),
     resolution - 1,
     max(date),
     round(avg(B01)),
     round(avg(B02)),
     round(avg(B03)),
     round(avg(B04)),
     round(avg(B05)),
     round(avg(B06)),
     round(avg(B07)),
     round(avg(B08)),
     round(avg(B09)),
     round(avg(B10)),
     round(avg(B11)),
     round(avg(B12)),
     round(avg(B8A)),
     (avg(B08) - avg(B04)) / (avg(B08) + avg(B04)),
     (avg(B11) - avg(B08)) / (avg(B11) + avg(B08)),
     (avg(B03) - avg(B08)) / (avg(B03) + avg(B08))
  FROM s2
  where resolution = 11
  and date = '2020-09-06 00:00:00'
  group by substring(zoneId, 1, length(zoneId) - 1), resolution
  having count(*) = 9

```

9.6. The ClickHouse DGGs data store

A second data store was written in GeoServer, reading data from ClickHouse tables satisfying the following structure:

- An string column named `zoneId`
- An integer column named `resolution`
- Any other attribute column

Similar to the geometry DGGs store, this store has no ties to a particular DGGs implementation, and requires the administration to declare which one to use when connecting to a database, in addition to common parameters such as database server host, port, database name, username and password.


ClickHouse DGGs integration
ClickHouse DGGs integration

Basic Store Info

Workspace *

dggs ▾

Data Source Name *

clickhouse-h3 

Description

Enabled

Connection Parameters

dggs_id *

H3

host *

localhost

port *

8123


database

act_h3

user *

default

passwd

..... 

Namespace *

https://www.geosolutionsgroup.com/tb16/dggs

Figure 9. Connecting to a ClickHouse server

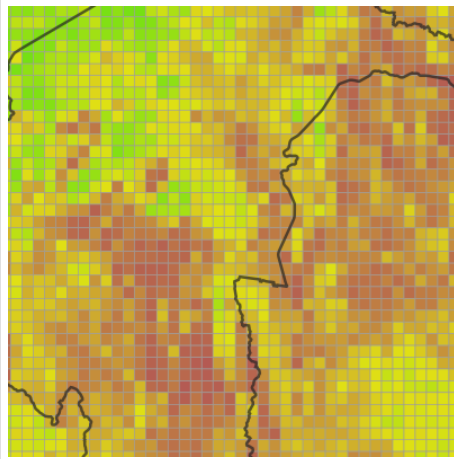
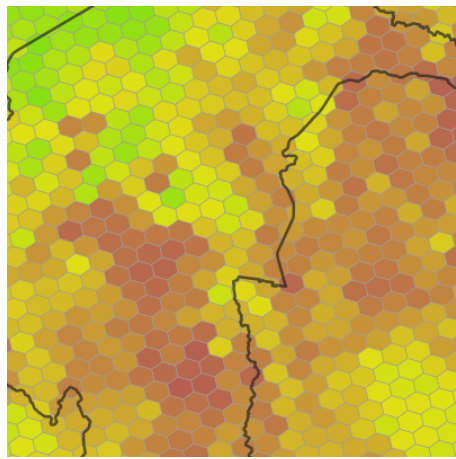
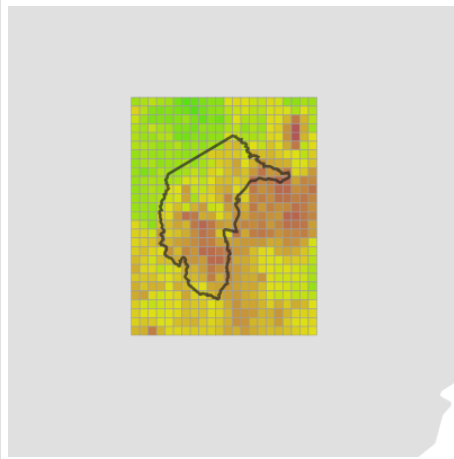
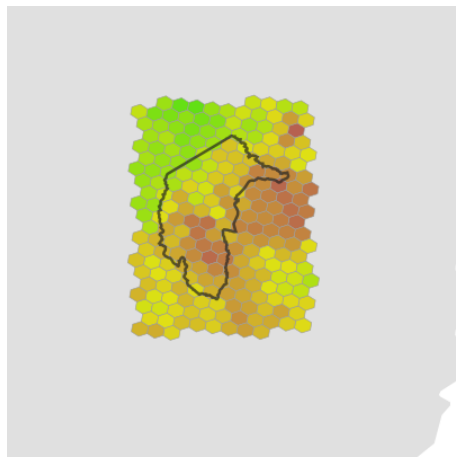
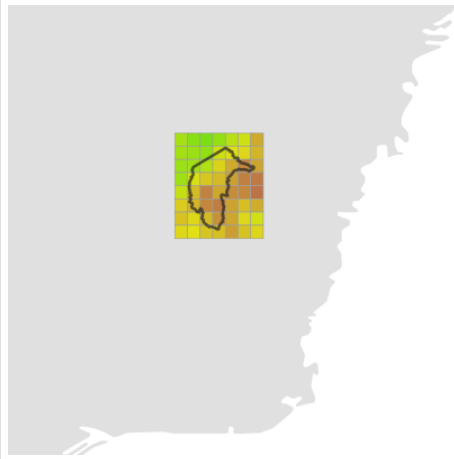
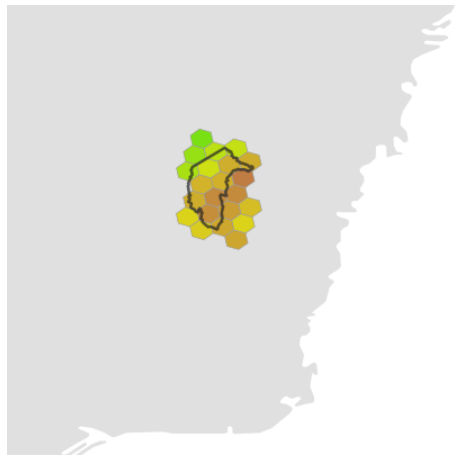
The store then makes available to GeoServer all tables matching the above conditions as feature collections, which can be configured as layers and consumed via the classic OGC services.

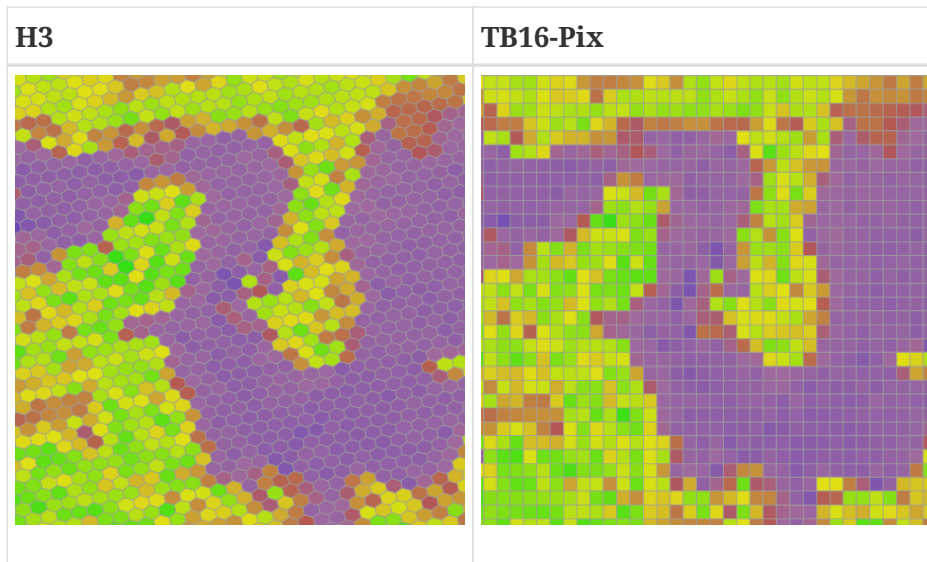
The figures in Table 38 visually compares H3 and TB16-Pix maps of the NDVI index over the test area, gently zooming in first, and then displaying a lake at the highest resolution. One of the two systems looks more fine-grained than the other. This is actually just an illusion, the issue being that the size of the zones in the two systems are not aligned.

Table 38. NDVI in H3 vs TB16Pix at progressively finer levels

H3

TB16-Pix





An interesting observation is to see how the store translates the bounding box request from a WMS endpoint into an efficient ClickHouse query. Here is a reference WMS request:

https://tb16.geo-solutions.it/geoserver/wms?STYLES=ndvi&LAYERS=dggs:s2-rpix&EXCEPTIONS=application%2Fvnd.ogc.se_inimage&TRANSPARENT=true&FORMAT=image%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&SRS=EPSG%3A4326&BBOX=148.69720458985,-35.911560058594,149.52117919923,-35.087585449219&WIDTH=600&HEIGHT=600

Table 39 shows the resulting rendering, and on the side, a map with the zone identifiers:

Table 39. WMS Rendering & Zone Ids

NDVI map	Zone identifiers																																																																
	<table border="1"> <tr> <td>P665221</td> <td>P673006</td> <td>P673007</td> <td>P673008</td> <td>P673016</td> <td>P673017</td> <td>P673018</td> <td>P673026</td> </tr> <tr> <td>P665252</td> <td>P673030</td> <td>P673031</td> <td>P673032</td> <td>P673040</td> <td>P673041</td> <td>P673042</td> <td>P673050</td> </tr> <tr> <td>P665255</td> <td>P673051</td> <td>P673034</td> <td>P673035</td> <td>P673043</td> <td>P673044</td> <td>P673045</td> <td>P673053</td> </tr> <tr> <td>P665258</td> <td>P673036</td> <td>P673037</td> <td>P673038</td> <td>P673046</td> <td>P673047</td> <td>P673048</td> <td>P673056</td> </tr> <tr> <td>P665282</td> <td>P673060</td> <td>P673061</td> <td>P673062</td> <td>P673070</td> <td>P673071</td> <td>P673072</td> <td>P673080</td> </tr> <tr> <td>P665285</td> <td>P673063</td> <td>P673064</td> <td>P673065</td> <td>P673073</td> <td>P673074</td> <td>P673075</td> <td>P673083</td> </tr> <tr> <td>P665288</td> <td>P673066</td> <td>P673067</td> <td>P673068</td> <td>P673076</td> <td>P673077</td> <td>P673078</td> <td>P673086</td> </tr> <tr> <td>P665522</td> <td>P673300</td> <td>P673301</td> <td>P673302</td> <td>P673310</td> <td>P673311</td> <td>P673312</td> <td>P673320</td> </tr> </table>	P665221	P673006	P673007	P673008	P673016	P673017	P673018	P673026	P665252	P673030	P673031	P673032	P673040	P673041	P673042	P673050	P665255	P673051	P673034	P673035	P673043	P673044	P673045	P673053	P665258	P673036	P673037	P673038	P673046	P673047	P673048	P673056	P665282	P673060	P673061	P673062	P673070	P673071	P673072	P673080	P665285	P673063	P673064	P673065	P673073	P673074	P673075	P673083	P665288	P673066	P673067	P673068	P673076	P673077	P673078	P673086	P665522	P673300	P673301	P673302	P673310	P673311	P673312	P673320
P665221	P673006	P673007	P673008	P673016	P673017	P673018	P673026																																																										
P665252	P673030	P673031	P673032	P673040	P673041	P673042	P673050																																																										
P665255	P673051	P673034	P673035	P673043	P673044	P673045	P673053																																																										
P665258	P673036	P673037	P673038	P673046	P673047	P673048	P673056																																																										
P665282	P673060	P673061	P673062	P673070	P673071	P673072	P673080																																																										
P665285	P673063	P673064	P673065	P673073	P673074	P673075	P673083																																																										
P665288	P673066	P673067	P673068	P673076	P673077	P673078	P673086																																																										
P665522	P673300	P673301	P673302	P673310	P673311	P673312	P673320																																																										

The resulting database query looks as follows:

```

SELECT "zoneId", "date", "NDVI"
FROM "act_rpix"."s2" WHERE
("date" = '2020-09-06 02:00:00.0' AND
 "zoneId" LIKE 'P67307%' AND "resolution" = 6) OR
 "zoneId" LIKE 'P67306%' AND "resolution" = 6) OR
 "zoneId" LIKE 'P67304%' AND "resolution" = 6) OR
 "zoneId" LIKE 'P67303%' AND "resolution" = 6) OR
 "zoneId" IN ('P673320', 'P673312', 'P673311', 'P673310', 'P673302',
 'P673301', 'P673300', 'P673086', 'P673083', 'P673080', 'P673056',
 'P673053', 'P673050', 'P673026', 'P673018', 'P673017', 'P673016',
 'P673008', 'P673007', 'P673006', 'P665522', 'P665288', 'P665285',
 'P665282', 'P665258', 'P665255', 'P665252', 'P665228')
)

```

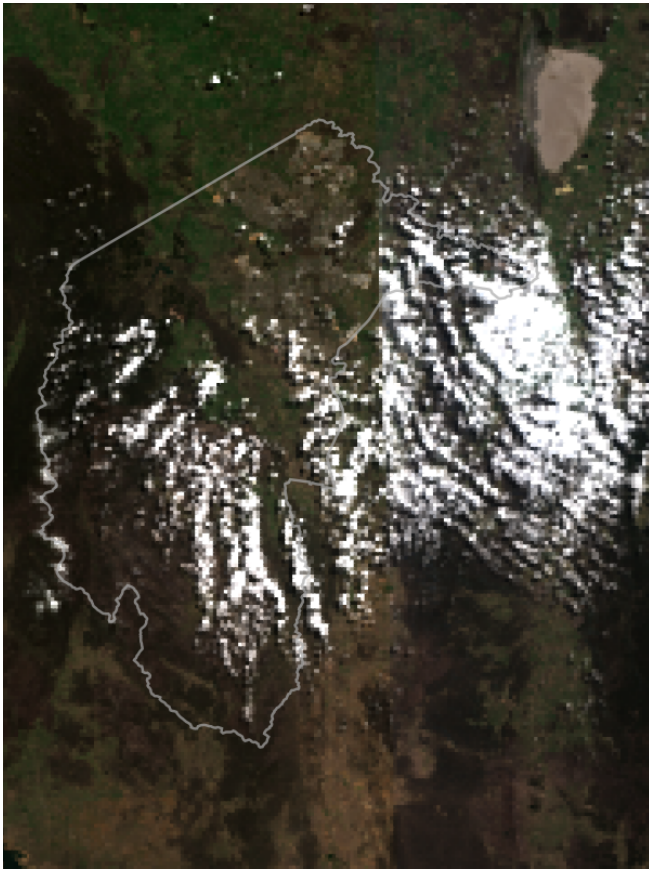
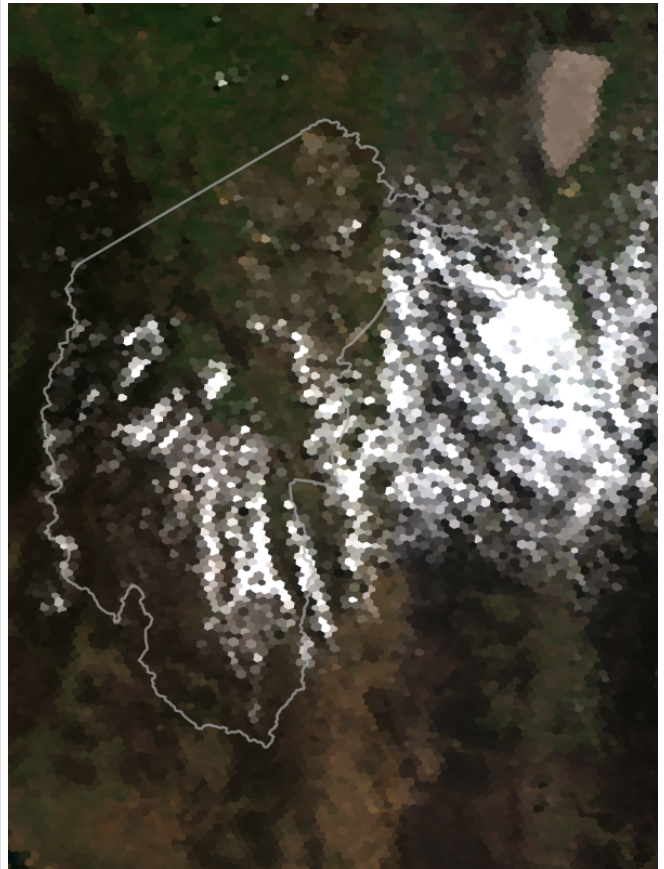
In other words, the system recognizes some of the zones are fully covered by their parent, and as a result, a query against the parent prefix identifier is done, selecting only the zones at the desired resolution.

Using the `viewparams=resoffset:1` vendor parameter to retrieve zones at resolution level 7 in the same area does not change the structure of the generated query, the system still recognizes P67307, P67307, P67304 and P67303 are covering most of the map. Therefore the "like" portion of the query is unchanged. However, the "in" portion of it has a different (and longer) list of zones partially intersecting the bounding box.

This approach helps keeping the queries small, which is particularly important when trying to render a large number of zones using the resolution offset vendor parameter.

9.7. Displaying false color maps of DGS data

Once the Sentinel 2 datasets have been loaded in GeoServer, it's possible to display false color maps of a given area. The following maps show a contrast stretched composition of B04,B03,B02 bands over the whole ACT area:

rHealPix**H3**

Focusing on Canberra, the following three maps show different views of the same area:

- A false color RGB image using NDBI,NDVI,NDWI, emphasizing built-up areas with the tones of red, vegetation areas with green, and water areas in blue. It's easy to identify water areas, built-up areas and roads, as well as a thick tree canopy covering the area, even in built-up areas. The datasets was not cloud masked during import, nor cloud-shadow masked, it's thus possible to locate both in the map as well.
- A false color RGB image using B04,B03,B02.
- A [GeoServer generated OpenStreetMap view](https://github.com/geosolutions-it/osm-styles) [https://github.com/geosolutions-it/osm-styles] of the area, acting as a visual reference to help interpretation of the other two maps

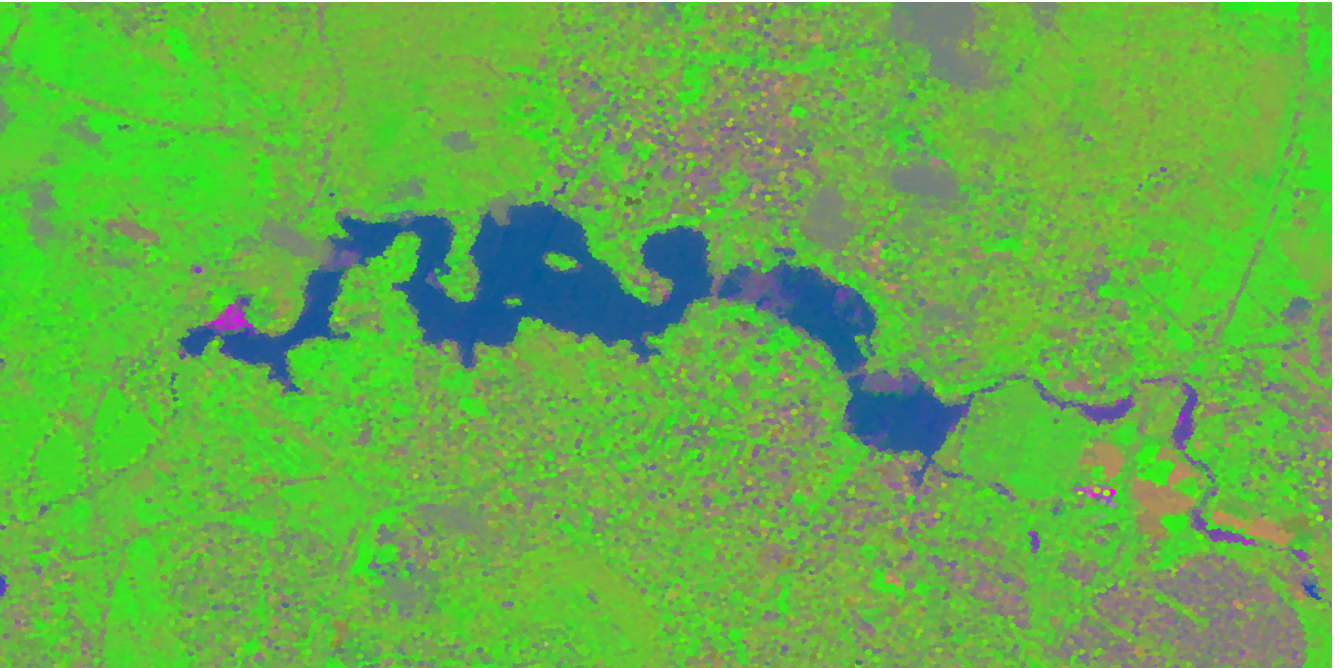


Figure 10. False color representation using NDBI,NDVI,NDWI bands, Canberra

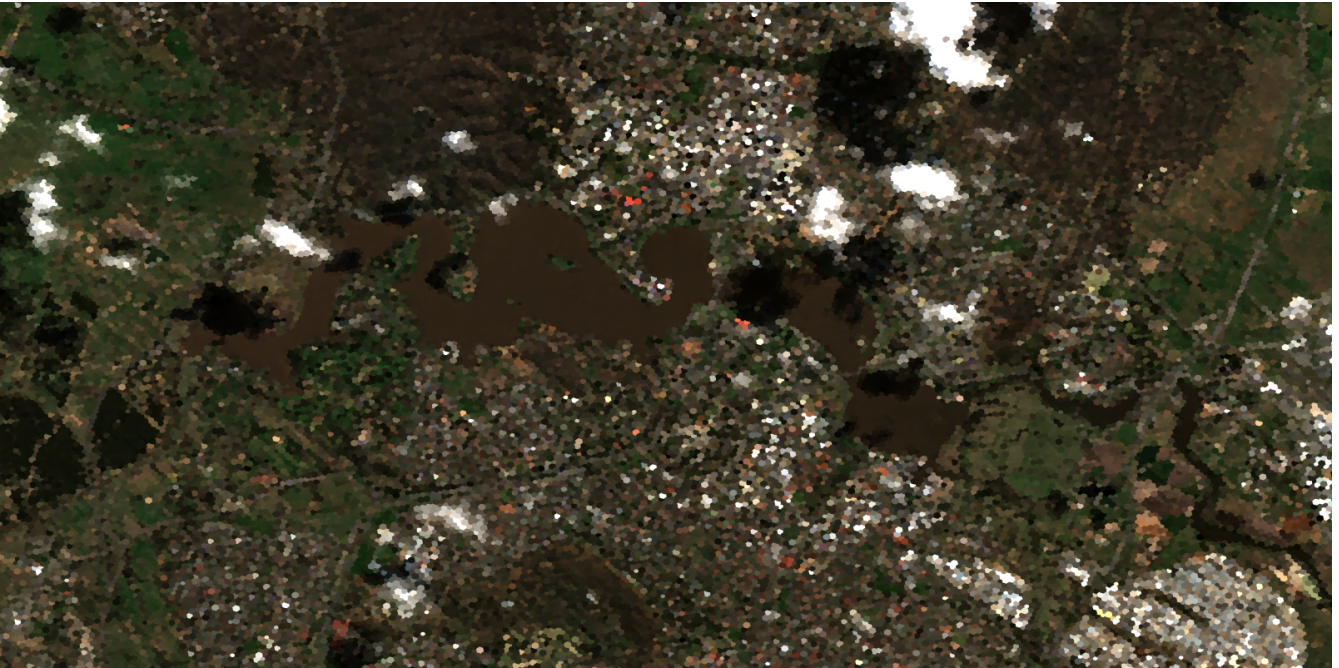


Figure 11. False color representation using B04,B03,B02 bands, Canberra

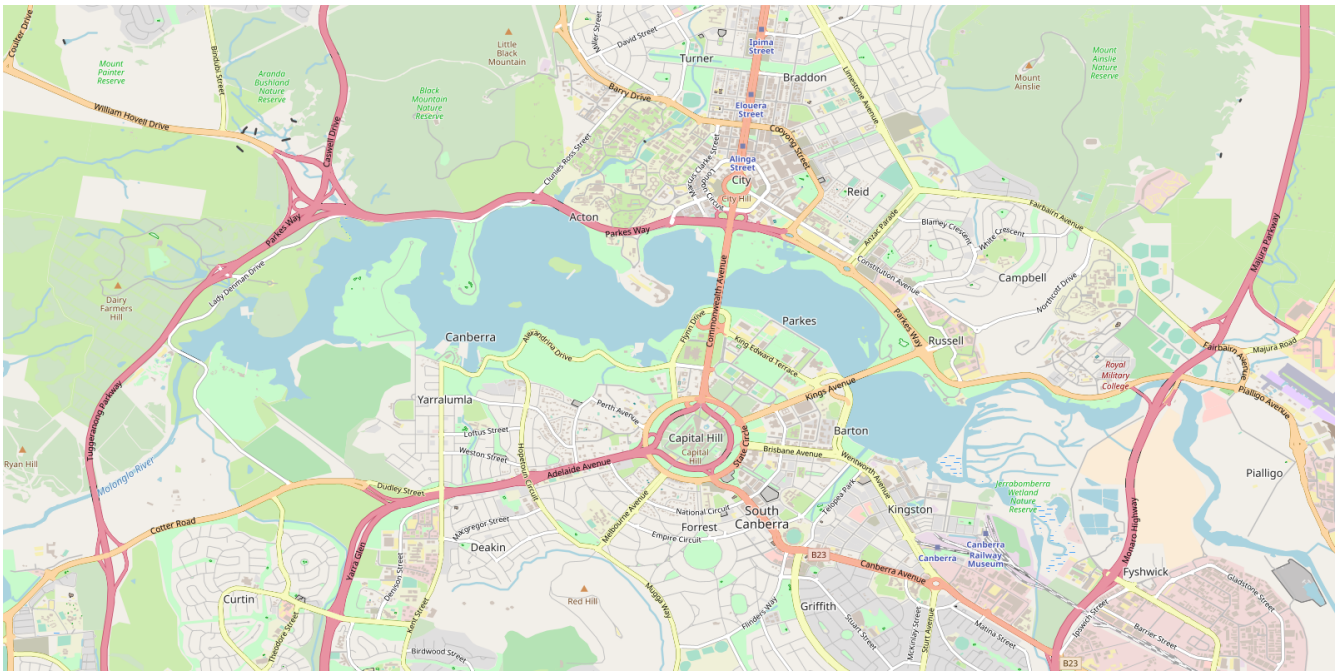


Figure 12. OSM reference map, Canberra

9.8. GeoServer DGGs API

The `ogcapi-dggs` module implements the process-oriented DGGs API described in the [DGGs process API](#) chapter.

The API is implemented using the GeoServer framework for OGC APIs, shared with OGC API - Features, and Styles and Tiles.

The implementation provides a HTML and JSON representation of most resources, with a few additional formats when required, or there is an opportunity for reuse of existing encoding support.

The following pictures provide a walk-through of the API resources available at the GeoSolutions Testbed-16 server: <https://tb16.geo-solutions.it/geoserver/dggs/ogc/dggs>

In particular, the pictures show the basics of the API as well as an exploration of the `TB16-Pix` collection, thereby providing a purely geometrical description of the TB16-Pix DGGs (backed by the `DGGGeometryStore` implementation).

GeoServer DGGs 1.0 Service

This is the landing page of the DGGs 1.0 service, providing links to the service API and its contents.
This document is also available as `application/x-yaml`, `application/json`, `application/cbor`.

API definition

The [API document](#) provides a machine processable description of this service API conformant to OpenAPI 3.
This API document is also available as `application/vnd.oai.openapi+json;version=3.0`, `application/x-yaml`, `application/cbor`, `text/html`.

Collections

The [collection page](#) provides a list of all the collections available in this service.
This collection page is also available as `application/x-yaml`, `application/json`, `application/cbor`.

Contact information

- Server managed by Andrea Aime
- Organization: GeoSolutions
- Mail: andrea.aime@geo-solutions.it

Figure 13. GeoServer DGGs API landing page

swagger <https://tb16.geo-solut> Explore

DGGS 1.0 server

2.19-SNAPSHOT OAS3

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/api?f=application%2Fvnd.oai.openapi%2Bjson%3Bversion%3D3.0>

Server

<https://tb16.geo-solutions.it/geoserver/ogc/dggs>

Capabilities

essential characteristics of this API

- GET / landing page
- GET /conformance information about specifications that this API conforms to.
- GET /collections the list of supported collections

DGGS access

- GET /collections/{collectionId} Describes a particular DGGS
- GET /collections/{collectionId}/zones Access the list of zones in a given DGGS. Can list either all the zones, or a particular subset based on resolution, WGS84 bbox, or list of containing zones (e.g., polygon defined in DGGS terms)
- GET /collections/{collectionId}/zone Access the definition of a particular zone
- GET /collections/{collectionId}/neighbors Get the list of neighboring zones, to a given zone (should it return just a list of identifiers instead of a GeoJSON collection? Could even be a list of links in the zone itself)

Figure 14. GeoServer DGGS API description, as a Swagger interactive HTML representation

GeoServer DGGS collections

This document lists all the collections available in the DGGS service.

dggs:H3

- **Title:** H3
- **Geographic extents:**
 - -180, -90, 180, 90.
- [DGGS zone listing.](#)

dggs:TB16-Pix

- **Title:** TB16-Pix
- **Geographic extents:**
 - -180, -90, 180, 90.
- [DGGS zone listing.](#)

dggs:s2-h3

- **Title:** s2
- **Geographic extents:**
 - -180, -90, 180, 90.
- [DGGS zone listing.](#)

dggs:s2-rpix

- **Title:** s2
- **Geographic extents:**
 - -180, -90, 180, 90.
- [DGGS zone listing.](#)

Figure 15. GeoServer DGGS API collections

TB16-Pix

fid	zoneld	resolution	shape	color
TB16-Pix.S	S	0	cap	#FF80FF
TB16-Pix.R	R	0	quad	#8080FF
TB16-Pix.Q	Q	0	quad	#80FFFF
TB16-Pix.P	P	0	quad	#80FF80
TB16-Pix.O	O	0	quad	#FFFF80
TB16-Pix.N	N	0	cap	#FF8080

Figure 16. TB16-Pix zone listing, at resolution 0. This is a purely geometrical collection, no actual data.

TB16-Pix

fid	zoneld	resolution	shape	color
TB16-Pix.S888888	S888888	6	dart	#FF8080
TB16-Pix.S888887	S888887	6	skew_quad	#FF8080
TB16-Pix.S888886	S888886	6	skew_quad	#FF8080
TB16-Pix.S888885	S888885	6	skew_quad	#FF8080
TB16-Pix.S888884	S888884	6	dart	#FF8080
TB16-Pix.S888883	S888883	6	skew_quad	#FF8080
TB16-Pix.S888882	S888882	6	skew_quad	#FF8080
TB16-Pix.S888881	S888881	6	skew_quad	#FF8080
TB16-Pix.S888880	S888880	6	dart	#FF8080
TB16-Pix.S888878	S888878	6	skew_quad	#FF8080
TB16-Pix.S888877	S888877	6	skew_quad	#FF8080
TB16-Pix.S888876	S888876	6	skew_quad	#FF8080
TB16-Pix.S888875	S888875	6	skew_quad	#FF8080
TB16-Pix.S888874	S888874	6	skew_quad	#FF8080
TB16-Pix.S888873	S888873	6	skew_quad	#FF8080

[Previous page](#) - [Next page](#)

Figure 17. GeoServer DGGs API TB16-Pix zone listing, at resolution 6. Notice the paging support.

Zone: S123456

- resolution: 6
- shape: skew_quad
- color: #FF80ED

Access zone relatives:

- [Zone parents](#)
- [Zone children](#)
- [Zone neighbor](#)

Figure 18. Inspecting a specific zone, S123456

TB16-Pix

fid	zonelid	resolution	shape	color
TB16-Pix.S12345	S12345	5	skew_quad	#FF80ED
TB16-Pix.S1234	S1234	4	skew_quad	#FF80ED
TB16-Pix.S123	S123	3	skew_quad	#FF80ED
TB16-Pix.S12	S12	2	skew_quad	#FF80EE
TB16-Pix.S1	S1	1	skew_quad	#FF80F1
TB16-Pix.S	S	0	cap	#FF80FF

Figure 19. Following the "zone parents" link, listing all parents of zone S123456

TB16-Pix

fid	zonelid	resolution	shape	color
TB16-Pix.S1234568	S1234568	7	skew_quad	#FF80ED
TB16-Pix.S1234567	S1234567	7	skew_quad	#FF80ED
TB16-Pix.S1234566	S1234566	7	skew_quad	#FF80ED
TB16-Pix.S1234565	S1234565	7	skew_quad	#FF80ED
TB16-Pix.S1234564	S1234564	7	skew_quad	#FF80ED
TB16-Pix.S1234563	S1234563	7	skew_quad	#FF80ED
TB16-Pix.S1234562	S1234562	7	skew_quad	#FF80ED
TB16-Pix.S1234561	S1234561	7	skew_quad	#FF80ED
TB16-Pix.S1234560	S1234560	7	skew_quad	#FF80ED

Figure 20. Following the "zone children" link, listing all direct children of zone S123456

TB16-Pix

fid	zonelid	resolution	shape	color
TB16-Pix.S123448	S123448	6	skew_quad	#FF80ED
TB16-Pix.S123480	S123480	6	skew_quad	#FF80ED
TB16-Pix.S123453	S123453	6	skew_quad	#FF80ED
TB16-Pix.S123457	S123457	6	skew_quad	#FF80ED

Figure 21. Following the "zone neighbors" link, listing all direct neighbors of zone S123456

What follows is a screenshot of the `s2-rpix` zone listing, containing Sentinel 2 data over the ACT, in September 2020:

s2-rpix

fid	zonelid	resolution	date	B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12	B8A	NDVI	NDBI	NDWI
s2-rpix.P665228.2020-09-06 00:00:00	P665228	6	6 set 2020, 02:00:00	125	260	496	471	2692	521	962	2162	2490	1803	1041	2745	2790	0.6420013499831253	-0.3498647430117223	-0.626629889669007
s2-rpix.P665252.2020-09-06 00:00:00	P665252	6	6 set 2020, 02:00:00	127	195	371	383	2287	470	783	1807	2102	1479	833	2339	2367	0.6504972599959407	-0.36896595261142207	-0.6592532136298714
s2-rpix.P665255.2020-09-06 00:00:00	P665255	6	6 set 2020, 02:00:00	83	163	293	300	1958	431	637	1520	1784	1096	573	2006	2024	0.6705121162180309	-0.4523455741880704	-0.6768580356595797
s2-rpix.P665258.2020-09-06 00:00:00	P665258	6	6 set 2020, 02:00:00	79	153	281	289	1830	382	621	1420	1657	1099	581	1871	1885	0.6616404523592877	-0.41930827735524345	-0.6696708463949843
s2-rpix.P665282.2020-09-06 00:00:00	P665282	6	6 set 2020, 02:00:00	77	208	332	371	1721	276	673	1349	1556	1178	682	1749	1767	0.5686781237886032	-0.3281549149390077	-0.6048648291360963
s2-rpix.P665285.2020-09-06 00:00:00	P665285	6	6 set 2020, 02:00:00	79	344	519	622	1956	201	958	1553	1756	1819	1123	1990	1996	0.4282209843103184	-0.16054817275747513	-0.4986861157290716
s2-rpix.P665288.2020-09-06 00:00:00	P665288	6	6 set 2020, 02:00:00	78	323	475	580	1659	178	853	1323	1490	1808	1256	1684	1703	0.39011149378320004	-0.02589066471373801	-0.4717260985105988
s2-rpix.P665522.2020-09-06 00:00:00	P665522	6	6 set 2020, 02:00:00	80	310	483	574	2014	183	917	1584	1804	1948	1292	2038	2050	0.46791636625811106	-0.10143359480659997	-0.5328027532802754
s2-rpix.P673006.2020-09-06 00:00:00	P673006	6	6 set 2020, 02:00:00	123	264	571	472	3610	582	1095	2855	3322	2036	1117	3619	3698	0.716432865731463	-0.43744230930603345	-0.6668288411014889
s2-rpix.P673007.2020-09-06 00:00:00	P673007	6	6 set 2020, 02:00:00	123	301	664	492	4372	662	1238	3446	4055	2180	1164	4382	4493	0.7501693002257337	-0.49510678301113614	-0.6767409169550173

[Previous page](#) - [Next page](#)

Figure 22. Following the "zone children" link, listing all direct children of zone S123456

The `s2-rpix` and `s2-h3` collections are interesting, as they are time enabled. Time management is configured in the UI as follows:

dgg:s2-rpix

Configure the resource and publishing information for the current layer

Data
Publishing
Dimensions
Tile Caching
Security

Time

Enabled

Attribute

End Attribute (Optional)

Presentation

Default value

Nearest Match

Figure 23. Configuring the time dimension in GeoServer

This affects both WMS instances and the DGGs API. In WMS the capabilities document reports the available times, and defaults to the September 2020 time slice.

GeoServer WMS 1.3 Capabilities excerpt

```

<Layer queryable="1" opaque="0">
  <Name>dgg:s2-rpix</Name>
  <Title>s2</Title>
  <Abstract/>
  <KeywordList>
    <Keyword>features</Keyword>
    <Keyword>s2</Keyword>
  </KeywordList>
  <CRS>EPSG:4326</CRS>
  <CRS>CRS:84</CRS>
  <EX_GeographicBoundingBox>
    <westBoundLongitude>-180.0</westBoundLongitude>
    <eastBoundLongitude>180.0</eastBoundLongitude>
    <southBoundLatitude>-90.0</southBoundLatitude>
    <northBoundLatitude>90.0</northBoundLatitude>
  </EX_GeographicBoundingBox>
  <BoundingBox CRS="CRS:84" minx="-180.0" miny="-90.0" maxx="180.0" maxy="90.0"/>
  <BoundingBox CRS="EPSG:4326" minx="-90.0" miny="-180.0" maxx="90.0" maxy="180.0"/>
  <Dimension name="time" default="2020-09-06T00:00:00Z" units="ISO8601">2019-09-02T00:00:00.000Z,2020-09-06T00:00:00.000Z</Dimension>
  <!-- Extra information omitted for brevity -->
</Layer>

```

At the DGGs API level, the time is exposed in the collection description:

All the API requests in addition can receive a `datetime` parameter specifying the desired time slice. If not specified, the API assumes the configured default.

The following request extracts a single zone, without a `datetime` specified, resulting in the September 2020 instance to be extracted:

https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/s2-h3/zone/?zone_id=86be0d207ffffff&f=application/dggs%2Bjson

Getting the 86be0d207ffffff H3 zone for the default time slice

```
{
  "type": "Feature",
  "geometry": {
    "type": "polygon",
    "identifiers": [
      "86be0d207ffffff"
    ]
  },
  "properties": {
    "resolution": 6,
    "date": "2020-09-06T00:00:00Z",
    "B01": 82,
    "B02": 1326,
    "B03": 1521,
    "B04": 1744,
    "B05": 2006,
    "B06": 576,
    "B07": 1926,
    "B08": 1892,
    "B09": 1988,
    "B10": 194,
    "B11": 126,
    "B12": 1850,
    "B8A": 868,
    "NDVI": 0.04062229904926536,
    "NDBI": -0.8752566371681416,
    "NDWI": -0.10884125920964499
  }
}
```

Providing a specific datetime allows to retrieve a different time slice:

```
{
  "type": "Feature",
  "geometry": {
    "type": "any",
    "identifiers": [
      "86be0d207ffffff"
    ]
  },
  "properties": {
    "resolution": 6,
    "date": "2019-09-02T00:00:00Z",
    "B01": 82,
    "B02": 965,
    "B03": 1142,
    "B04": 1365,
    "B05": 2020,
    "B06": 712,
    "B07": 1580,
    "B08": 1735,
    "B09": 1869,
    "B10": 3104,
    "B11": 2605,
    "B12": 2081,
    "B8A": 2102,
    "NDVI": 0.11926267281105993,
    "NDBI": 0.2005792903692976,
    "NDWI": -0.20607806137650211
  }
}
```

All API endpoints returning zones accept a `datetime` query parameter, allowing full usage of temporal datasets.

The following example retrieves resolution 6 zones matching the CRS84 point `149,-35` during September 2019:

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/s2-h3/point?point=149,-35&resolution=6&datetime=2019-09&f=application/dggs%2Bjson>

```
{
  "type": "Feature",
  "geometry": {
    "type": "any",
    "identifiers": [
      "86be7258ffffff"
    ]
  },
  "properties": {
    "resolution": 6,
    "date": "2019-09-02T00:00:00Z",
    "B01": 88,
    "B02": 434,
    "B03": 719,
    "B04": 808,
    "B05": 3020,
    "B06": 680,
    "B07": 1323,
    "B08": 2403,
    "B09": 2750,
    "B10": 2861,
    "B11": 1871,
    "B12": 3090,
    "B8A": 3145,
    "NDVI": 0.4967305724834305,
    "NDBI": -0.12463651860022062,
    "NDWI": -0.5394610422290342
  }
}
```

Similarly, a polygon mapping operation can be performed, mapping the CRS84 ellipsoidal triangle to a set of resolution 5 zones, for September 2019:

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/s2-h3/polygon?polygon=POLYGON149%20%20-36,%20149%20-35,%20150%20-35,%20149%20-36,%20&resolution=5&datetime=2019-09&f=application/dggs%2Bjson>

Polygon mapping, with attributes other than NDVI omitted, as well as several features, to reduce the example size

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "polygon",
        "identifiers": [
```

```

        "85be0d27ffffff"
    ]
  },
  "properties":{
    "resolution":5,
    "date":"2019-09-02T00:00:00Z",
    "NDVI":0.34315113754366094
  }
},
{
  "type":"Feature",
  "geometry":{
    "type":"polygon",
    "identifiers":[
      "85be0d27ffffff"
    ]
  },
  "properties":{
    "resolution":5,
    "date":"2020-09-06T00:00:00Z",
    "NDVI":0.5679273827534039
  }
},
{
  "type":"Feature",
  "geometry":{
    "type":"polygon",
    "identifiers":[
      "85be0d37ffffff"
    ]
  },
  "properties":{
    "resolution":5,
    "date":"2019-09-02T00:00:00Z",
    "NDVI":0.32704305233307546
  }
},
  "Other features omitted for brevity"
],
"numberMatched":20,
"numberReturned":20,
"timeStamp":"2020-10-15T10:31:39.382Z",
"Links omitted for brevity": null
}

```

9.9. GeoServer DGGs based DAPA API

During Testbed-16 GeoSolutions implemented a DAPA API based on the DGGs data stored in ClickHouse, taking advantage of the Lightweight Directory Access Protocol (LDAP) database fast aggregation abilities.

9.9.1. The API, HTML representations, and process resources

In particular, the implementation included the "area" and "position" resources, while the "grid" ones have been skipped due to development time limitations:

DAPA	
GET	<code>/collections/{collectionId}/processes</code> list the available data retrieval patterns
GET	<code>/collections/{collectionId}/variables</code> fetch the observable properties included in this observation collection
GET	<code>/collections/{collectionId}/processes/area:retrieve</code> retrieve a time series for selected variables for each s
GET	<code>/collections/{collectionId}/processes/area:aggregate-space</code> retrieve a time series for selected variable
GET	<code>/collections/{collectionId}/processes/area:aggregate-time</code> retrieve a time series for selected variables
GET	<code>/collections/{collectionId}/processes/area:aggregate-space-time</code> retrieve a time series for selected
GET	<code>/collections/{collectionId}/processes/position:retrieve</code> retrieve a time series for selected variables at t
GET	<code>/collections/{collectionId}/processes/position:aggregate-time</code> retrieve a time series for selected va

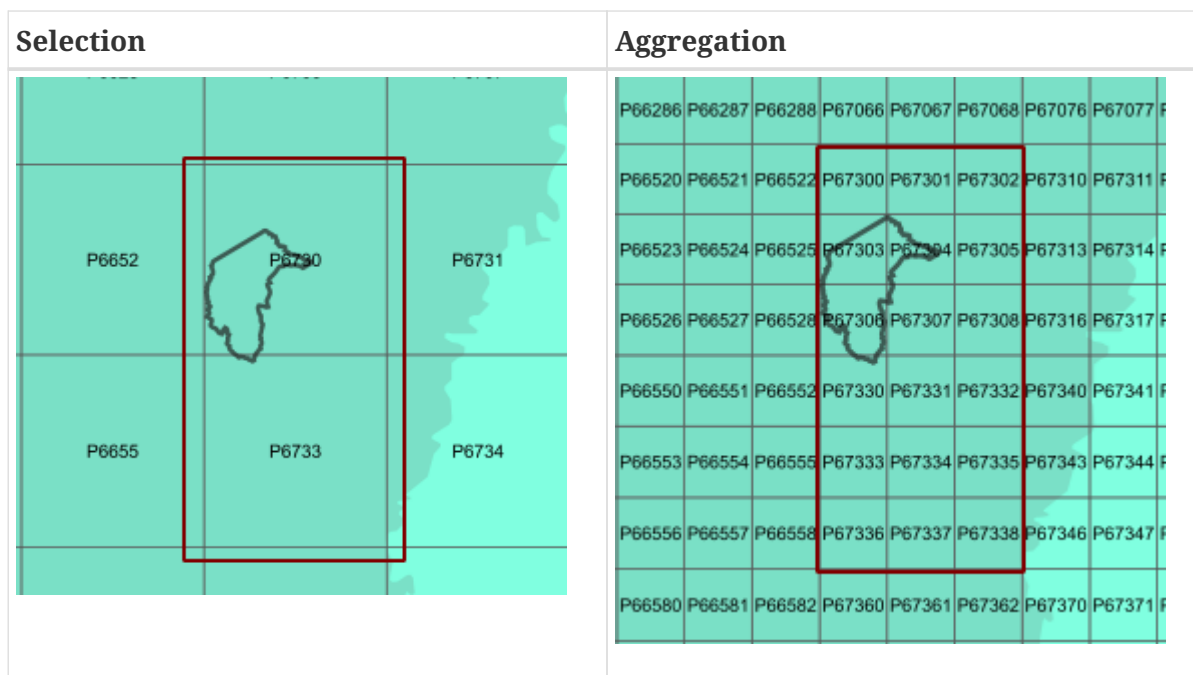
Figure 24. DAPA API extensions

In addition to the DAPA space selection mechanisms, that is, bounding box and polygon, this DGGs inspired API also allows to specify a list of comma-separated zone identifiers in the "zones" parameter. When zones are specified, the data retrieval or aggregations are performed inside the area covered by the zones:

`zones=P6730,P6733`

In addition to that, a `resolution` parameter is present, which allows to specify the target resolution for the retrieval or aggregation operations. When the zones listed in the `zones` parameter have a resolution lower than the aggregation one, their children at the target resolution level are used to run the aggregation instead.

Table 40. Zones used for selection versus aggregation, when using `zones=P6730,P6733&resolution=5` in the request



The description of DAPA enabled collections has been extended with links to the DAPA subsystem, thus linking the DGGS and DAPA resources:

[dggs:s2-h3](#)

- **Title:** s2
- **Geographic extents:**
 - -180, -90, 180, 90.
- [DGGS zone listing.](#)
- [DAPA processes.](#)
- [DAPA variables.](#)

[dggs:s2-rpix](#)

- **Title:** s2
- **Geographic extents:**
 - 148.76, -35.92, 149.4, -35.12.
- [DGGS zone listing.](#)
- [DAPA processes.](#)
- [DAPA variables.](#)

Figure 25. Collections listing with DAPA links

The `processes` endpoint provides a full listing of aggregation functions, variables for the collection:

dggs:s2-rpix DAPA support

The following endpoints are available to retrieve and process the dggs:s2-rpix zones in addition to the standard DGGs queries. The endpoints are described in the API definition and the links point to the specification of the operation in the OpenAPI definition with the available input parameters and the response schema

Variables

- B01: Field of type Short
- B02: Field of type Short
- B03: Field of type Short
- B04: Field of type Short
- B05: Field of type Short
- B06: Field of type Short
- B07: Field of type Short
- B08: Field of type Short
- B09: Field of type Short
- B10: Field of type Short
- B11: Field of type Short
- B12: Field of type Short
- B8A: Field of type Short
- NDVI: Field of type Double
- NDBI: Field of type Double
- NDWI: Field of type Double

Functions

- min
- max
- mean
- count
- sum
- std-dev

Figure 26. DAPA processes resource, variables and functions

The `processes` resource also provides a list of all available methods of data retrieval and aggregation:

Processes

The following processes are supported on `dggs:s2-rpix`.

`area:retrieve`

This DAPA endpoint returns observation values at the selected location (parameter `coord` or `coordRef`) in the selected time interval or at the selected time instant (parameter `datetime`).

This DAPA endpoint returns `dggs:s2-rpix` values for an area (parameter `bbox`, `geom` or `zones`) in the selected time interval or at the selected time instant (parameter `datetime`).

No aggregation is performed, this just returns the data as-is.

Run "`area:retrieve`" with default parameters

`area:aggregate-space`

Retrieve a time series for selected variables for each station in an area and apply functions on the values of each time step.

This DAPA endpoint returns a time series for an area in the selected time interval `dggs:s2-rpix` values for an area (parameter `bbox`, `geom` or `zones`) in the selected time interval or at the selected time instant (`datetime`).

All values in the area for each requested variable (parameter `variables`) are aggregated for each time step and each of the requested statistical functions (parameter `functions`) is applied to the aggregated values

Run "`area:aggregate-space`" with default parameters

`area:aggregate-time`

Retrieve a time series for selected variables for each zone in an area and apply functions on the values of each time series.

This DAPA endpoint returns a time aggregate for each zone in an area, in the selected time interval.

Each result contains contains the aggregation functions evaluated over the time series of each value associated to the zone.

Run "`area:aggregate-time`" with default parameters

`area:aggregate-space-time`

Retrieve a time series for selected variables for each station in an area and apply functions on all values.

This DAPA endpoint returns `dggs:s2-rpix` values for an area (parameter `bbox`, `geom` or `zones`) in the selected time interval or at the selected time instant (parameter `datetime`).

All values for each requested variable (parameter `variables`) are aggregated and each of the requested statistical functions (parameter `functions`) is applied to the aggregated values.

Run "`area:aggregate-space-time`" with default parameters

`position:retrieve`

This DAPA endpoint returns a time series in the selected zone (or point), in the selected time interval or at the selected time instant (parameter `datetime`).

The time series contains values for each selected variable (parameter `variables`) for which a value can be interpolated at the location.

Run "`position:retrieve`" with default parameters

`position:aggregate-time`

This DAPA endpoint returns values at the selected zone (parameter `geom` or `zone_id`) in the selected time interval or at the selected time instant (parameter `datetime`).

All values in the time interval for each requested variable (parameter `variables`) are aggregated and each of the requested statistical functions (parameter `functions`) is applied to the aggregated values.

Run "`position:aggregate-time`" with default parameters

Figure 27. DAPA processes resource, available processes with default parameter links

The `area:retrieve` method returns a time series of values in a given area (or the whole area, if not specified). No aggregation is performed in this case, making it a close match to the DGGs API `zones` resource.

Here is an example URL, and the associated result:

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/area:retrieve?resolution=5&f=text/csv&datetime=2020>

The result, showing the four zones available at resolution level 5 over ACT, with data for year 2020 (only one time slice available)

```
FID,zoneId,resolution,date,B01,B02,B03,B04,B05,B06,B07,B08,B09,B10,B11,B12,B8A,NDVI,ND
BI,NDWI,geometry
s2-rpix.P67303.2020-09-06 00:00:00,P67303,5,2020-09-
06T02:00:00,96,456,660,667,2704,519,1108,2186,2499,1854,1155,2740,2784,0.53248676,-
0.3083907,-0.53613682,"POLYGON ((-35.10748095969024 148.74999999999997,
-35.10748095969024 149.12037037037038, -35.49268851161001 149.12037037037038,
-35.49268851161001 148.74999999999997, -35.10748095969024 148.74999999999997))"
s2-rpix.P67304.2020-09-06 00:00:00,P67304,5,2020-09-
06T02:00:00,85,1592,1691,1701,3370,587,2110,2906,3157,2792,2147,3341,3636,0.26166241,-
0.15013853,-0.26437671,"POLYGON ((-35.10748095969024 149.12037037037038,
-35.10748095969024 149.49074074074076, -35.49268851161001 149.49074074074076,
-35.49268851161001 149.12037037037038, -35.10748095969024 149.12037037037038))"
s2-rpix.P67306.2020-09-06 00:00:00,P67306,5,2020-09-
06T02:00:00,81,859,977,1067,2089,319,1326,1789,1947,2217,1742,2120,2230,0.25279172,-
0.01334424,-0.29330013,"POLYGON ((-35.49268851161001 148.74999999999997,
-35.49268851161001 149.12037037037038, -35.879718636556554 149.12037037037038,
-35.879718636556554 148.74999999999997, -35.49268851161001 148.74999999999997))"
s2-rpix.P67307.2020-09-06 00:00:00,P67307,5,2020-09-
06T02:00:00,80,1200,1274,1330,2687,466,1662,2290,2490,2538,1948,2682,2947,0.26502548,-
0.0806587,-0.28482354,"POLYGON ((-35.49268851161001 149.12037037037038,
-35.49268851161001 149.49074074074076, -35.879718636556554 149.49074074074076,
-35.879718636556554 149.12037037037038, -35.49268851161001 149.12037037037038))"
```

The `area:aggregate-time` takes every zone in the area of interest, and performs the selected aggregation functions on each of the variables over all values found in the time slices available.

Here is an example URL, and the associated result:

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/area:aggregate-time?resolution=5&f=text/csv&variables=B01,B02,B03&functions=min,mean,max,count>

Aggregating the 4 TB16Pix zones covering ACT, at resolution 5, over the two time slices available.

```
FID,geometry,zoneId,B01_min,B01_average,B01_max,B01_count,B02_min,B02_average,B02_max,
B02_count,B03_min,B03_average,B03_max,B03_count
area_time_P67307,"POLYGON ((148.76 -35.92, 149.4 -35.92, 149.4 -35.12, 148.76 -35.12,
148.76 -35.92))",P67307,80,80,80,2,559,879.5,1200,2,713,993.5,1274,2
area_time_P67303,"POLYGON ((148.76 -35.92, 149.4 -35.92, 149.4 -35.12, 148.76 -35.12,
148.76 -35.92))",P67303,85,90.5,96,2,329,392.5,456,2,530,595,660,2
area_time_P67304,"POLYGON ((148.76 -35.92, 149.4 -35.92, 149.4 -35.12, 148.76 -35.12,
148.76 -35.92))",P67304,83,84,85,2,558,1075,1592,2,783,1237,1691,2
area_time_P67306,"POLYGON ((148.76 -35.92, 149.4 -35.92, 149.4 -35.12, 148.76 -35.12,
148.76 -35.92))",P67306,80,80.5,81,2,332,595.5,859,2,471,724,977,2
```

The `area:aggrate-space` takes every time slice, and aggregates the variables of each zone in the area of interest instead. The following example operates at resolution 9, aggregating for each variable over 260 thousand zones (executes in around five seconds):

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/area:aggregate-space?bbox=148.7,-35.9,149.4,-35.0&resolution=10&f=text/csv&variables=B01,B02,B03&functions=min,mean,max,count>

Aggregating over 260k zones per time slice, in the requested bounding box, producing one result for each of the time slices available.

```
FID,geometry,date,B01_min,B01_average,B01_max,B01_count,B02_min,B02_average,B02_max,B02_count,B03_min,B03_average,B03_max,B03_count
area_space_time_2020-09-06_02:00:00.0,"POLYGON ((148.7 -35.9, 149.39999999999998 -35.9, 149.39999999999998 -35, 148.7 -35, 148.7 -35.9))",2020-09-06T02:00:00,63,89.12817284,151,261280,1,810.27268065,17610,261280,1,960.22063686,16776,261280
area_space_time_2019-09-02_02:00:00.0,"POLYGON ((148.7 -35.9, 149.39999999999998 -35.9, 149.39999999999998 -35, 148.7 -35, 148.7 -35.9))",2019-09-02T02:00:00,60,83.28399801,115,261280,1,429.43291105,13476,261280,1,613.64177893,14225,261280
```

The `area-aggregate-space-time` process takes every zone in the area of interest, for every time in the selected range, and produces global statistics, resulting in a single record instead. Here is an example, with a compact `datetime` filter selecting both available years:

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/area:aggregate-space?bbox=148.7,-35.9,149.4,-35.0&resolution=10&f=text/csv&variables=B01,B02,B03&functions=min,mean,max,count>

Aggregating all 500 thousand zones available in the selected bounding box, at resolution 9, into a single space-time summary result.

```
FID,geometry,phenomenonTime,B01_min,B01_average,B01_max,B01_count,B02_min,B02_average,B02_max,B02_count,B03_min,B03_average,B03_max,B03_count
space-time-aggregate,"POLYGON ((148.7 -35.9, 149.39999999999998 -35.9, 149.39999999999998 -35, 148.7 -35, 148.7 -35.9))",2019/2020,60,86.20608543,151,522560,1,619.85279585,17610,522560,1,786.9312079,16776,522560
```

In addition to `area`, the DAPA implementation offers `position` data retrieval and aggregation as well. The `position` data retrieval and aggregation, concentrates on a given point, making it a close relative to the DGGs API `zone` endpoint. Similar to the DGGs API, the `position` implementation in GeoServer also accepts a `zone_id` to identify a particular zone. As an alternative, it is possible to specify a single point and a resolution, which will be mapped to the unique zone containing the point at the given resolution.

<http://localhost:8080/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/position:retrieve?>

[resolution=9&f=text%2Fcsv&geom=149.08%2C-35.52&datetime=2020](http://localhost:8080/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/position:aggregate-time?zone_id=P673062236&resolution=9&f=text%2Fcsv&geom=149.08%2C-35.52&datetime=2020)

Retrieving data for a specific point at resolution 9, matching a TB16Pix zone

```
FID,zoneId,resolution,date,B01,B02,B03,B04,B05,B06,B07,B08,B09,B10,B11,B12,B8A,NDVI,ND
BI,NDWI,geometry
s2-rpix.P673062236.2020-09-06 00:00:00,P673062236,9,2020-09-
06T02:00:00,88,492,762,748,3318,667,1380,2835,3223,2639,1532,3505,3573,0.58257364,-
0.29844557,-0.57641463,"POLYGON ((-35.51652603409146 149.07921810699588,
-35.51652603409146 149.08379058070415, -35.52129437280328 149.08379058070415,
-35.52129437280328 149.07921810699588, -35.51652603409146 149.07921810699588))"
```

The `position:aggregate-time` aggregates the values of the target variables, in the selected zones, over all the times available in the time series:

http://localhost:8080/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/position:aggregate-time?zone_id=P673062236&resolution=9&f=text%2Fcsv&datetime=2019/2020&variables=NDVI,NDBI,NDWI&functions=min,max,std-dev

Retrieving data for a specific zone, and aggregating it over the available time series

```
FID,geometry,zoneId,NDVI_min,NDVI_max,NDVI_std-dev,NDBI_min,NDBI_max,NDBI_std-
dev,NDWI_min,NDWI_max,NDWI_std-dev
position_time_P673062236,POINT (149.08150434385
-35.518910203447376),P673062236,0.34747217,0.58257364,0.11755074,
-0.29844557,0.03754498,0.16799527,-0.57641463,-0.4686294,0.05389261
```

9.9.2. Notes on implementation and performance

An efficient implementation of DAPA should try to perform all of the aggregations, on all of the variables, in a single ClickHouse SQL request. This is indeed how GeoServer implements it, there are however significant performance differences in computation based on how the type of area filtering is chosen by the client.

The simplest and most efficient query derives from a client that uses DGGs zone references to identify the area of interest. Given the following request:

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/area:aggregate-space-time?zones=P6730,P6733&resolution=11&f=text/csv&variables=B01&functions=min,mean,max,count>

the server has to locate all zones at resolution 11 whose parent is either `P6730` or `P6733`, and aggregate the results. Given the hierarchical nature of TB16Pix identifiers, the query is as simple as:

```
SELECT min("B01"), avg("B01"), max("B01"), count(*)
FROM "act_rpix"."s2"
WHERE (("zoneId" LIKE 'P6730%' OR "zoneId" LIKE 'P6733%' )
AND "resolution" = 11)
```


Clickhouse runs this query against 8 million records in 200ms.

If the client instead uses a generic bounding box, not aligned to the DGGS grid, then the shape needs to be approximated, using a mix of lower-level zones and higher resolution zones, up to the target resolution level.

The following request, limited to resolution 7 in order to reduce the size of the query, results in the following:

<http://localhost:8080/geoserver/ogc/dggs/collections/dggs:s2-rpix/processes/area:aggregate-space-time?bbox=148.7,-35.9,149.4,-35.0&resolution=7&f=text/csv&variables=B01&functions=min,mean,max,count>

Results in the following SQL query, with **like** operators used to match zones by their parent, and a **in** operator matching the zones at the target resolution:

```
SELECT min("B01"), avg("B01"), max("B01"), count(*)
FROM "act_rpix"."s2"
WHERE (("zoneId" LIKE 'P673006%' OR "zoneId" LIKE 'P673007%' OR "zoneId" LIKE
'P673008%' OR "zoneId" LIKE 'P673016%' OR "zoneId" LIKE 'P673017%' OR "zoneId" LIKE
'P67303%' OR "zoneId" LIKE 'P673040%' OR "zoneId" LIKE 'P673041%' OR "zoneId" LIKE
'P673043%' OR "zoneId" LIKE 'P673044%' OR "zoneId" LIKE 'P673046%' OR "zoneId" LIKE
'P673047%' OR "zoneId" LIKE 'P67306%' OR "zoneId" LIKE 'P673070%' OR "zoneId" LIKE
'P673071%' OR "zoneId" LIKE 'P673073%' OR "zoneId" LIKE 'P673074%' OR "zoneId" LIKE
'P673076%' OR "zoneId" LIKE 'P673077%'
OR "zoneId" IN ('P6652281', 'P6652282', 'P6652284', 'P6652285', 'P6652287', 'P6652288
', 'P6652521', 'P6652522', 'P6652524', 'P6652525', 'P6652527', 'P6652528', 'P6652551',
'P6652552', 'P6652554', 'P6652555', 'P6652557', 'P6652558', 'P6652581', 'P6652582',
'P6652584', 'P6652585', 'P6652587', 'P6652588', 'P6652821', 'P6652822', 'P6652824',
'P6652825', 'P6652827', 'P6652828', 'P6652851', 'P6652852', 'P6652854', 'P6652855',
'P6652857', 'P6652858', 'P6652881', 'P6652882', 'P6652884', 'P6652885', 'P6652887',
'P6652888', 'P6655221', 'P6655222', 'P6730180', 'P6730183', 'P6730186', 'P6730420',
'P6730423', 'P6730426', 'P6730450', 'P6730453', 'P6730456', 'P6730480', 'P6730483',
'P6730486', 'P6730720', 'P6730723', 'P6730726', 'P6730750', 'P6730753', 'P6730756',
'P6730780', 'P6730783', 'P6730786', 'P6733000', 'P6733001', 'P6733002', 'P6733010',
'P6733011', 'P6733012', 'P6733020', 'P6733021', 'P6733022', 'P6733100', 'P6733101',
'P6733102', 'P6733110', 'P6733111', 'P6733112', 'P6733120'))
AND "resolution" = 7)
```

The maps in [Table 41](#) show visually the construction of the above query for TB16Pix, and offers a parallel for the H3 DGGS as well. In red the requested bounding box, in blue the ACT area, in dark gray the zones used to build the database query for resolution 7.

Table 41. Query construction with parent and child features in both TB16Pix and H3, at resolution levels 7 and 8, for the bounding box 148.7, -35.9, 149.4, -35.0 (ACT boundary shown for reference only).

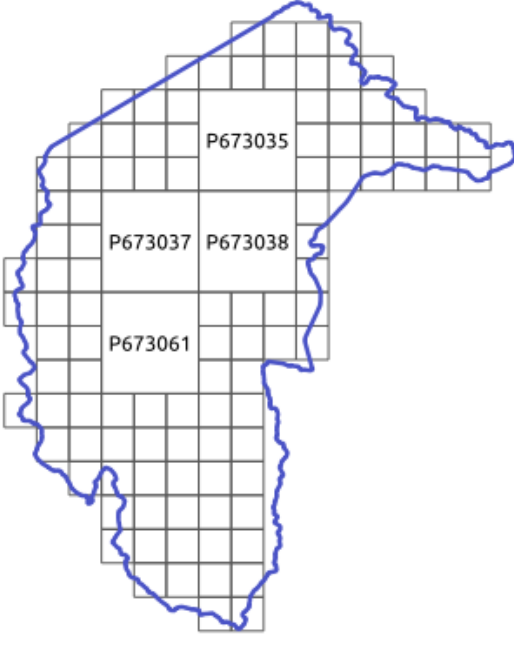

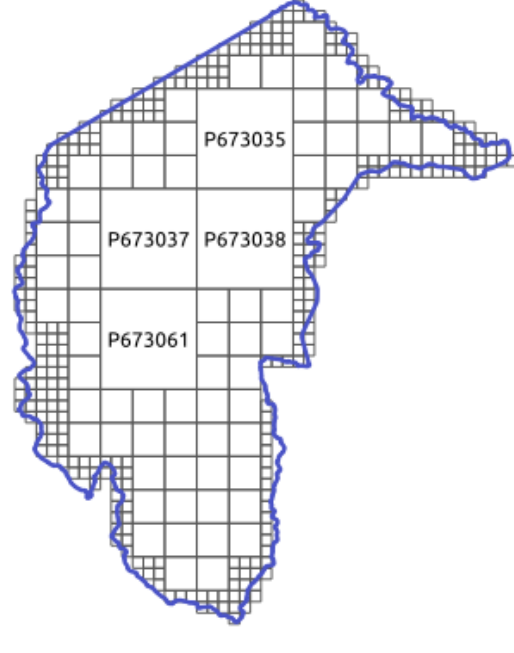

Resolution	TB16Pix	H3
7		
8		

Trying to perform the same aggregation at resolution 11 generates thousands of SQL conditions, resulting in a significant amount of time spent generating the query, in the database parsing it and deciding on an execution plan, and eventually in its execution. In particular, the same request at resolution 11 requires over a minute of computation time, despite aggregating only 4.7 million zones, while the previous request based on zones aggregated over 8 million zones in just 0.2 seconds.

For reference, the same happens when the area of aggregation is a polygon. The example in [Table 42](#) shows the zones used for querying the area covered by ACT itself:

Table 42. Query construction with parent and child features in both TB16Pix and H3, at resolution

levels 7 and 8, for a simplified version of the ACT polygon.

Resolution	TB16Pix	H3
7		
8		

In future implementations it might be interesting to allow specifying two different resolutions, a target resolution for aggregation, and a resolution used to approximate the area of interest as a list of DGGs zones. This is important as the lower resolution levels of a DGGs data set might have been computed using average, while the statistic of interest might be the minimum and the maximum, which would be lost by querying the lower resolution levels.

[15] DGGs zones can be converted into a multi-band raster using the appropriate space filling curve, and then stored, for example, as tiled, compressed TIFFs.

[16] Reference system used a AMD Ryzen 1700x, 8 core, 32GB of RAM, 512GB of NvME SSD storage. It is believed that, to reach this performance, the actual data was cached in memory by either the database or the operating system.

Chapter 10. DGGs Demo Client

OpenWork Ltd, on behalf of Manaaki Whenua Landcare Research delivered the **D138 DGGs Demo Client** - Client application with DGGs API support and capable of demonstrating DGGs capabilities. These open-source clients provide visualization of DGGs encoded data based on the TB16Pix DGGs RS, at various zoom levels. They also interact with D139 DGGs-enabled data services with an OGC API endpoint that understands ZoneIDs as spatial filters. Preliminary capability to consume data from D137 DGGs Server Implementation was also demonstrated. Viewer code is available at <https://git.nzoss.org.nz/openwork/pydggin>.

10.1. Background - Choice to create a native DGGs viewer

The decision was made early to develop a DGGs native viewer. While extending an existing map-viewer was considered, the participants felt that to do so would limit demonstrating advantages that DGGs provides potential new data viewing platforms. While using tools such as Leaflet to view DGGs data is possible, doing so would undermine DGGs's simplicity. This is because there is no need in DGGs for viewers to address the complexity of projecting coordinate pairs onto a flat screen. Support for conversion of coordinate pair geometries is central to existing map-viewers, but not required for DGGs. The hope is that creation of a native viewer in this testbed would illustrate how simple and lightweight such a viewer could be.

The problem the participants wished to avoid was making DGGs seem harder to use than it is. Retrofitting existing coordinate-pair based viewers to also support DGGs data requires a heavier platform with more code. This approach would not illustrate the simplicity of DGGs nor the lightweight viewing platforms believed possible when using ZoneIDs as geometry. The participants wished to illustrate how using ZoneIDs as geometry could allow the creation of tools that are more lightweight and easier to develop than those based on a coordinate pair paradigm.

The viewers developed demonstrate the ability to view DGGs data only for rHEALPix data whose base geometry is square. This simplified viewer development for the purposes of the Testbed. This same approach could be applied to the two DGGs geometries (hexagons and triangles). The mathematics involved in mapping the zones to a screen would be somewhat more complex but not unreasonably so. The participants recommend this be addressed in future testbeds.

10.2. Theory and logic behind the native DGGs viewers

What does DGGs native mean?

1. Use of ZoneIDs to describe both location and geometry.
2. ZoneIDs also provide geographic index.
3. No requirement for use coordinate pairs to describe geometry.

Theoretic advantages of DGGs native?

1. Speed and simplicity:

- a. Because there is no need to support [scan line \(vector\) algorithms](https://en.wikipedia.org/wiki/Scanline_rendering) [https://en.wikipedia.org/wiki/Scanline_rendering].
 - b. Lightweight – can run on very small platforms and large variety of devices.
 - c. Proven scalability.
2. Translating data to the screen is easier:
- a. At least within a single face.
 - b. Less distance between the thing and its geographic description – no calculation required in order to know where to place on screen.
3. Easier to link location data:
- a. ZoneIDs can be used as true identifiers in RDF.
 - b. Geometries are described by sets of ZoneIDs that naturally translate into RDF.
 - c. Preservation of scale – no zero-dimension points.
4. Easier to perform spatial analysis:
- a. Set theory rather than scan line (vector) algorithms simplifies mathematics.
 - b. The lack of zero-dimension points simplifies DE-9IM algorithms.

10.2.1. Examples

Table 43. EXAMPLE - description of Vector Analysis types and related DGGS equivalents

Type	Precision	Operation	Performance	Special requirements
BBox overlay	Low - limited to minimum bounding box in projected space	Geometric	Fast	Common projection
	High as desired - based on footprint of ZoneIDs	Set theory	Veryfast	Common DGGS RS
Ringfence	High - but unknown (0 dimension points)	Scan line (vector) algorithms	Data, Software and hardware dependent - at least loglinear	GIS software
	Variable - but known (DGGS zone area)	Set theory	Fast and Constant	Common DGGS RS

Type	Precision	Operation	Performance	Special requirements
Overlay analysis	Variable - unknown (0 dimension points)	Scan line (vector) algorithms	Data, Software and hardware dependent - at least loglinear	GIS software
	Variable - but known (DGGS zone area)	Set theory	Fast and at most Linear	Common DGGS RS

10.3. Client implementation

Building the client went through four iterations:

1. The first client was a C program that rendered multi-resolution DGGS geometries cropped to a given ancestor zone ID. The zones in this viewer may be assigned a color individually or collectively. The viewer parses input to an attributed trie, selects a subtrie, and uses a trivial recursive function to convert that to an image via [Cairo Vector Graphics](https://www.cairographics.org/) [https://www.cairographics.org/].
2. An interactive viewer wrapped this C program to enable navigating between DGGS zones using the keyboard. This showed only one ZoneID onscreen, with the arrow keys triggering it to recompute which ZoneID that should be by zooming in/out or navigating up/down/left/right.
3. For a nicer interactive viewer a tiled map viewer enabling pan and zoom using the mouse was created. This calls the C renderer for more complicated geometry. This updated a central ZoneID and its screen size to respond to mouse gestures.
4. To demonstrate geospatial analysis capabilities in DGGS clients, a fourth viewer was created that integrates into Jupyter notebooks. This essentially works the same as the first viewer, but with added DE-9IM operators that can be called from Python. The DE-9IM operators are reimplemented directly upon the [trie](https://en.wikipedia.org/wiki/Trie) [https://en.wikipedia.org/wiki/Trie] in Python, saving a format conversion and effort of exposing our existing DGGS DE-9IM analysis library to Python.

10.4. Viewers

1. [PyDGgin.py](https://git.nzoss.org.nz/openwork/pydggin/-/blob/master/pydggin.py) [https://git.nzoss.org.nz/openwork/pydggin/-/blob/master/pydggin.py] - A python desktop tool for viewing DGGS data. Meets the requirements of the testbed to pan, zoom and identify.
 - a. [dggs-render \(render.c\)](https://git.nzoss.org.nz/openwork/pydggin/-/blob/master/render.c) [https://git.nzoss.org.nz/openwork/pydggin/-/blob/master/render.c] - C library to render and process rHEALPix DGGS data. Utilised by Pydggin.
2. [Jupyter notebook DGGS viewer](https://git.nzoss.org.nz/openwork/pydggin/-/blob/master/render.py) [https://git.nzoss.org.nz/openwork/pydggin/-/blob/master/render.py] - To demonstrate the ability of a DGGS client to do spatial calculations on TB16Pix DGGS data.
 - a. reimplements the dggs-render and [libDGGS](https://git.nzoss.org.nz/openwork/libdggs) [https://git.nzoss.org.nz/openwork/libdggs] libraries in python.

10.4.1. How do the viewers work? Paint by numbers

1. Zone-ids (and their corresponding geometries) are placed on the display based on a parent zone-id of a given pixel size.
2. This process is repeated recursively for all the zone's children and in turn grandchildren, etc.
3. Multi-resolution DGGs geometries are rendered and cropped to a given ancestor zone ID.
4. Location of parents, children, grandchildren, etc. are also known and predetermined.
5. Data other than image tiles are reformatted into a common "Indental" format.
6. Data is parsed from these Indental files into a trie for the rendering process to traverse.

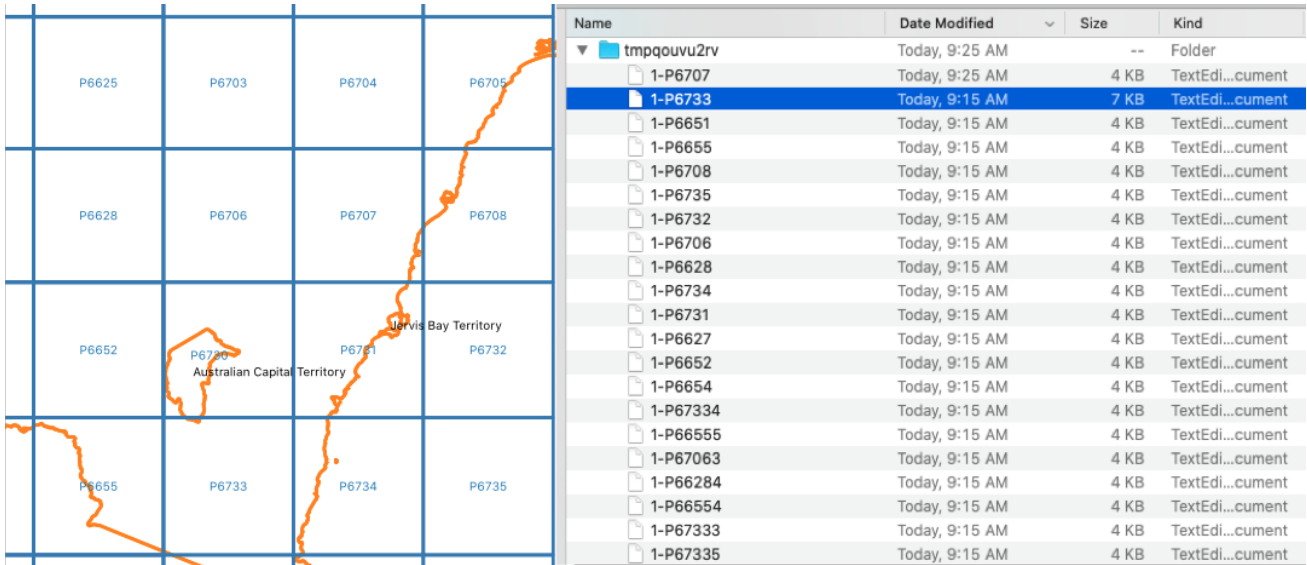


Figure 28. DGGs Level 5 Grid

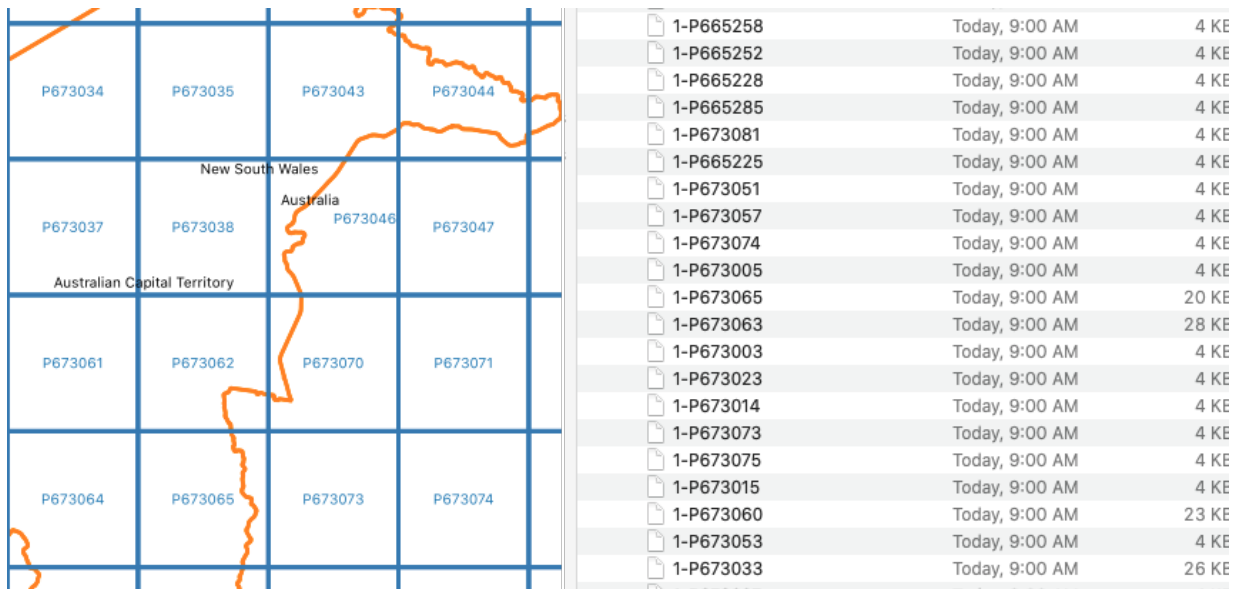


Figure 29. DGGs Level 7 Grid

10.4.2. PyDGGin.py

PyDGGin.py is a desktop native DGGs navigator written in Python. It currently only supports

rHealPIX DGGs RS. Future work is recommended to support additional DGGs RS. PyDGGin relies on custom the C libraries in render.c for composing tiled imagery.

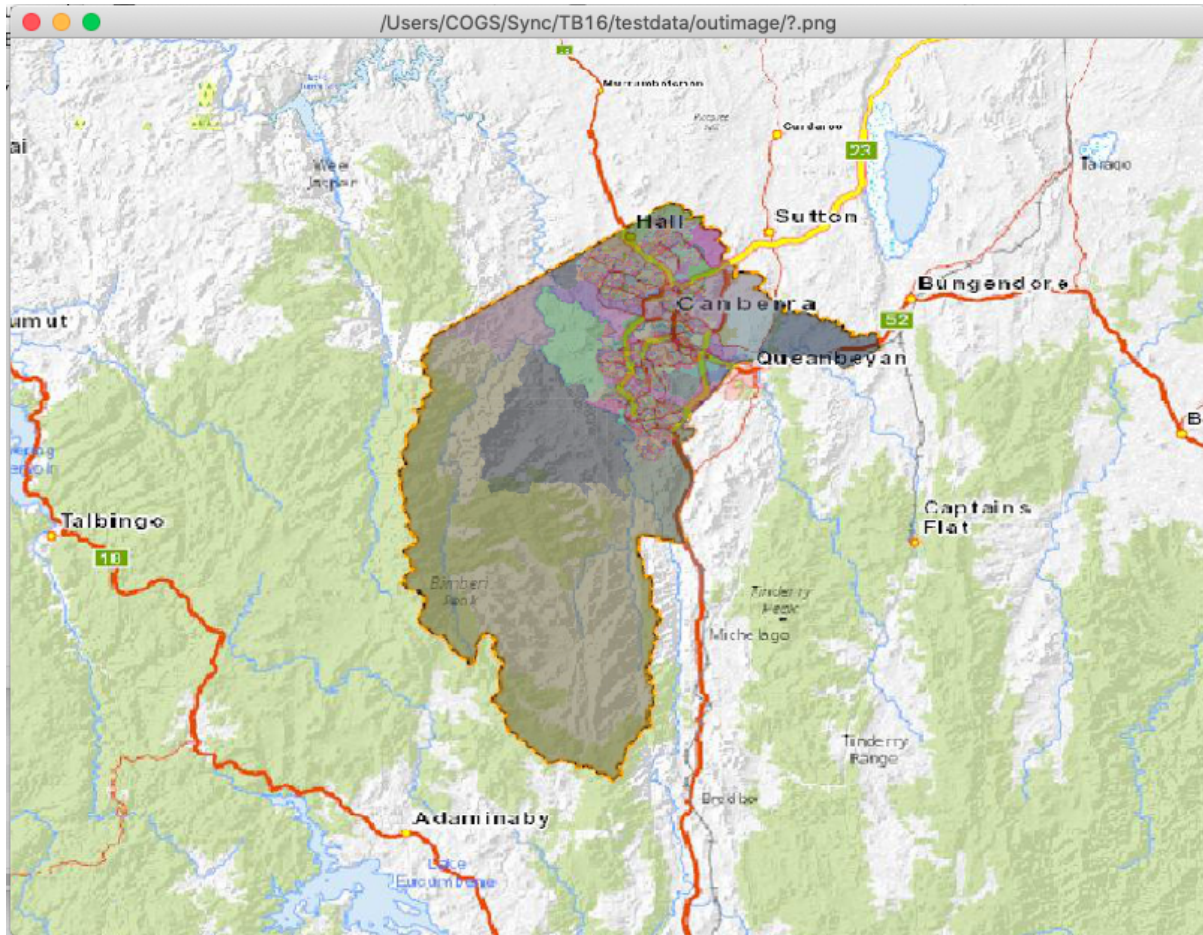


Figure 30. PyDGGin View Window

```

(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer %
(base) COGS@OWLminiServer dggs-viewer % python pydggin.py "/Users/COGS/Sync/TB16/testdata/outimage/? .png" 1.indental -P67303
P67304
P6730377 {'label': 'https://linked.data.gov.au/dataset/asgs2016/statisticalarealevel1/80111114002', 'red': '0.157137777963675', 'green': '0.18705
655851293512', 'blue': '0.23908036599250992', 'alpha': '0.5'}
P67303

```

Figure 31. PyDGGin Command Line

Command line arguments in PyDGGin determine which layers are loaded and in what order. A central ZoneID prefixed with a "-" determines the starting position and zoom level. Styling is predetermined in the code. Clicking on a location returns details about the data and ZoneIDs to the command line. There is no ability to interact with the data or interface beyond identify, pan and zoom using a mouse.

The original version of PyDGGin was built using the [pygame](https://www.pygame.org/) [https://www.pygame.org/] libraries. Later the participants migrated to [Pyglet](http://pyglet.org/) [http://pyglet.org/] to take advantage of greater hardware optimizations (batch render). It was also assumed that it might help with some transparency issues but those turned out to be caching issues instead.

To provide and update the display in PyDGGin, the central ZoneID, coordinate offset, and tile size are tracked and updated by mouse events. After these events all the other onscreen ZoneIDs are computed and composited to render the view. An early attempt tried to convert from a single layer-grid to a multi-layer grid rather than track the central ZoneID, but that turned out to be especially "buggy" when dealing with the poles or when zooming.

10.4.2.1. Challenges

The main difficulties were in figuring out how to use the graphics libraries effectively to composite layers together. Issues specific to DGGs were less problematic.

The original challenge was how to display an indexed DGGs fabric on the screen. To simplify things, the testbed participants only tested support for TB16Pix. The first issue, which proved relatively easy to resolve was nested tessellation. Recursion based on the properties of the space filling curve indexing of TB16Pix was easy to translate into code.

The next issue, provision of a slippy map interface, proved to be somewhat more difficult. Traversing the primary faces of the base polyhedron was a challenge - especially when tiling across equatorial to polar faces. While not demonstrated here a solution was devised. This requires further testing and development. There were a few major issues when navigating across a relatively flat plane of a single face.

Lack of native DGGs data was a major barrier to development of the viewer. Reliance on converted traditional GIS data created significant overhead. Some conversion issues by relatively young and untested existing tools created further complications which required some manual interventions. Development and use of DGGs native data formats in future testbeds is recommended. DGGs data issues are more deeply discussed in the next section.

1. Other Miscellaneous Issues – PyDGGin

- a. Projection translation issues – greatest at poles with "dart" and "trapezoidal" zones (when viewed in WGS 84 or similar).
- b. Performance – speed and display issues.
 - i. Most issues likely relate to unfamiliarity with graphics tools.
 - ii. Slow load times from web services.
- c. Styling improvements needed.
- d. No tool bars and info boxes – currently.

2. Render.c

- a. Simple recursive rendering routine parses data formats into a trie. Requires optimization.
 - i. Currently specific to rHealPIX. Requires refactoring testing to support other DGGs RS.
 - ii. Other DGGs RS can have their own recursive routines with details to suit.
 - iii. Formats – current formats done for convenience - need further work in future Testbeds.

10.5. Jupyter Notebook DGGs

In addition to PyDGGin a second viewer integrated into Jupyter Notebook was developed. This second viewer went beyond the requirements of this Testbed and demonstrates DGGs analysis capability by supporting native DGGs DE-9IM operators. Testing to date has been very limited. This viewer provides a good platform for more development in future Testbeds.

Jupyter notebook incorporates the same principles as most of the other renderers, outputting a PNG image for Jupyter Notebooks to display. The DGGs DE-9IM operations are performed directly on the same trie data-structure used for rendering.

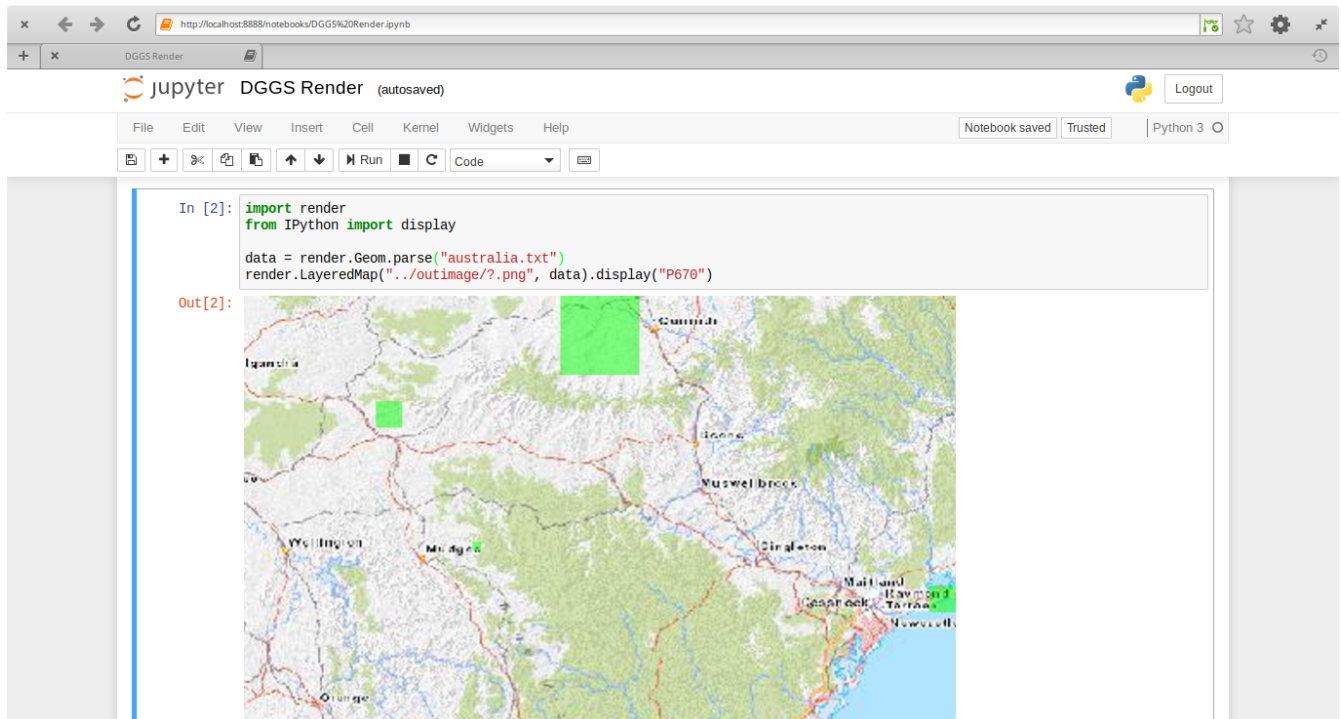


Figure 32. Jupyter Notebook DGGs

10.6. Data

PyDGGin and Jupyter Notebook DGGs both support multiple formats for DGGs data.

For initial data an indentation-based ".indental" format listing features and their (indented) ZoneIDs was used. Indented beneath either of those can be specified rendering parameters and other properties.

For a backdrop, an argument of a template filepath (ending in ".png") to a directory of image files named for each ZoneID was accepted. PyDGGin will replace the "?" with the ZoneID for which it needs an image. PyDGGin retrieves matches from this directory and populates the viewer with these. For the purposes of the TB16Pix DGGs clients, existing rHEALPix libraries to convert coordinate pair data to DGGs were used. [ScenzGrid-py](https://github.com/manaakiwhenua/scenzgrid-py) [https://github.com/manaakiwhenua/scenzgrid-py] was also utilized to create a mosaic of image tiles as a backdrop using rHEALPix libraries. Initially some discrepancies were found in the processes for conversion of coordinate pairs to zone-ids. These were resolved by tiling the original WGS 84 data rather than the re-projected version that the

ScenzGrid-py created. This worked in this instance because the projection for this face of the TB16Pix was also WGS84. The error would need correcting if we are to use the ScenzGrid-py software for other situations.

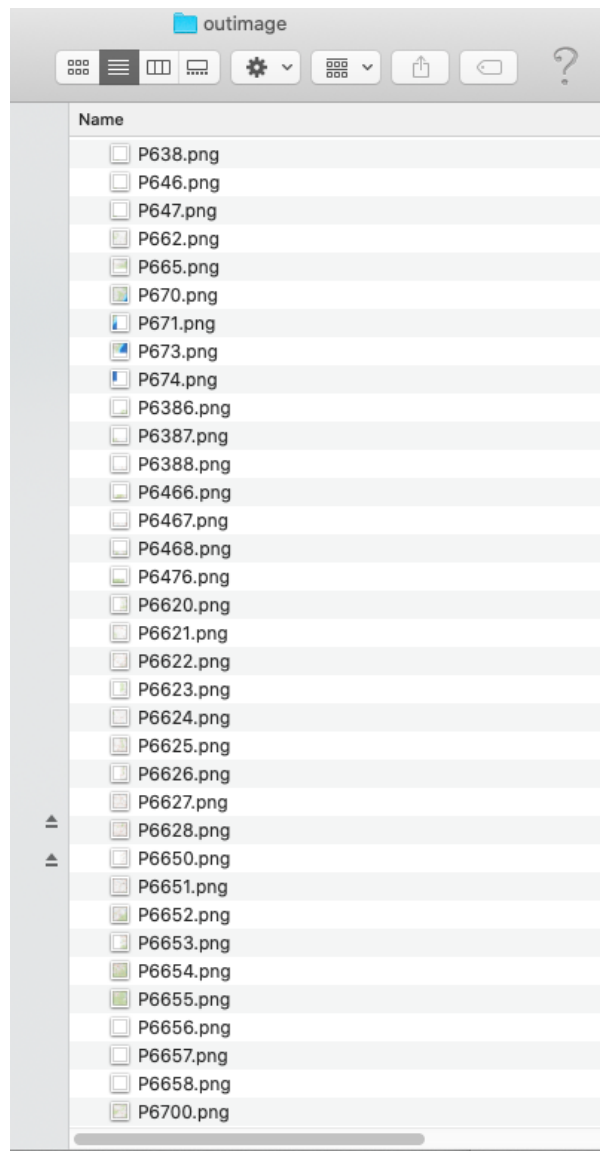


Figure 33. Structured Image Directory contents

The second format implemented was a CSV format with at least 2 columns. The first column was a name for the feature, while the second had space-separated ZoneIDs. For rendering this was pre-processed into an ".indental" file with consistent rendering parameters.

```
1.indental
https://linked.data.gov.au/dataset/asgs2016/statisticalarealevel1/80103103201
  label: https://linked.data.gov.au/dataset/asgs2016/
statisticalarealevel1/80103103201
  red: 0.6927595126056552
  green: 0.31078291384419654
  blue: 0.5266634429677766
  alpha: 0.5
P67304358882
P67304358884
P67304358885
P67304358886
P67304358887
P67304358888
P67304382212
P67304382214
P67304382215
P67304382216
P67304382217
P67304382218
P6730438222
P6730438224
P6730438225
P6730438227
P6730438228
P67304382235
P67304382237
P67304382238
P67304382261
P67304382262
P67304382264
```

Figure 34. ".indental" File for CSV or Http from D139

Then support for more complex JSON & RDF-based formats added. This was translated into ".indental". Both the RDF-based and JSON-based formats were supported as downloads from a webservice. Sample Sentinel 2 data was provided by the GeoSolutions delivered D137 DGGS Server Implementation as JSON.

```
1.indental
geom0
  label: act_rpix_2019_09_r9.json
  red: 0.05217391304347826
  green: 0.053012967200610224
  blue: 0.029214340198321892
  alpha: 0.5
  resolution: 9
  date: 2020-09-06T00:00:00Z
  B01: 121
  B02: 383
  B03: 695
  B04: 684
  B05: 3074
  B06: 391
  B07: 1315
  B08: 2516
  B09: 2820
  B10: 2536
  B11: 1536
  B12: 3122
  B8A: 3113
  NDVI: 0.5727083333333334
  NDBI: -0.24191823640700832
  NDWI: -0.5672122071900626
  P665224574
geom1
  label: act_rpix_2019_09_r9.json
  red: 0.0738367658276125
  green: 0.06903127383676583
  blue: 0.0418001525553013
  alpha: 0.5
  resolution: 9
  date: 2020-09-06T00:00:00Z
  B01: 120
  B02: 548
  B03: 905
  B04: 968
  B05: 2995
  B06: 378
  B07: 1581
  B08: 2501
  B09: 2765
  B10: 2884
  B11: 1822
  B12: 3051
  B8A: 3055
  NDVI: 0.44180385625520463
  NDBI: -0.15698570987971622
  NDWI: -0.46847170119067033
  P665224575
geom2
  label: act_rpix_2019_09_r9.json
  red: 0.02387400465203660
```

Figure 35. ".indental" File for Sentinel 2 JSON data

This Testbed work proved the ability to consume live feeds of RDF data from SURROUND’s D139 OGC API - Feature oriented API service. JSON feeds from GeoSolutions, while yet to be made available, were successfully tested on sample output data.

An immediate issue for building DGGs native viewers is the dearth of DGGs data available. Until such a time that spatial data is captured with zone-ids as geometry, there will be a need to re-project traditional GIS data to a DGGs RS suitable for the viewer. The participants recommend that future DGGs testbeds include DGGs data capture clients which use DGGs zone-ids for location and geometry.

Any tool is only as good as the data that goes into it. As conversion between coordinate systems is a fundamental principle of GIS, it is important that these conversion tools be as robust and fool-proof

as possible. Further development and testing of such tools to support a full range of DGGS RS should be a focus of future work.

DGGS Data Format

For the purposes of development, the ".indental" format was created using TB16Pix data to test the viewer. This format was used to hold a variety of different data, both real and mock. While it seems suitable as a temporary file format, its ability to hold wide variety of data is yet to be tested. As shown above, these ".indental" files hold names, attributes, styles and geometries (as ZoneIDs). Therefore, this is considered to be very much an alpha format that could be used for data exchange. Future work is needed to test and refine DGGS data exchange formats.

Data Types

The DGGS viewers were designed to demonstrate the ability to share multiple types of data using a DGGS to describe location. The types of data demonstrated in the viewers include tiled imagery as backdrop, points as single ZoneIDs, and polygons as multiple ZoneIDs. One possibility is that for many cases, pre-conversion of existing GIS data to DGGS formats, or the capture and storage of data in DGGS formats is the most efficient solution.

10.7. DGGS API Queries

As per the Testbed-16 CFP, one of the purposes of **D138 DGGS Demo Client** is to test the utility of APIs developed in **D139** and to a lesser extent, **D137**. However, to support DGGS viewers only a small subset of the API functionality provided by these servers proved to be needed. The DGGS specific API parameters required by these viewers were limited to *zones* (or *bounding box*) to allow a filter by one or more Zone IDs, and *resolution* to limit the returned data to a particular DGGS level. Other more generic API parameters used supported the MIME type and profile selection and paging functionality.

In testing the **D138** clients, data was retrieved from two different APIs. The first **D139** delivered feature type data, catchments and statistical area boundaries for the Australian Capital Territory. An example URI for this service was:

http://asgs.surroundaustralia.com/collections/SA1s/items?per_page=40&page=3&bbox=P6730

The following header information was included in this request:

```
_HEADERS = {  
  "Accept": "text/turtle",  
  "Accept-Profile": "<http://www.opengis.net/ont/geosparql>",  
  "User-Agent": "PyDGGin"  
}
```

The second API provided Sentinel 2 image data for the same region. This was developed by GeoSolutions as part of their **D137** DGGS Server API development requirements. This request was based on the request `GET /collections/{collectionId}/zones` as described previously. An example URI for this service was:

<https://tb16.geo-solutions.it/geoserver/ogc/dggs/collections/dggs:s2-rpix/zones?f=application/>

[dggs+json&limit=5000&resolution=8&zones=P6730,P6652,P6651,P6731&startIndex=5000](#)

No header information was included in this request.

The differences between these APIs from the client perspective is summarized in the table below:

Table 44. Comparison of two DGGS APIs

Query Type	Surround Ltd	example	GeoSolutions	example	Notes
Zone Filter	<i>bbox</i> [0..1]	P6730	<i>zones</i> [0..*]	P6730,P6652, P6651	
Resolution	<i>na</i>		<i>resolution</i>	7	Not required for data from Surround server
MIME type	<i>Accept</i> (header info)	"text/turtle"	<i>f</i>	application/dggs+json	GeoSolutions combined MIME type and Profile in a custom MIME type
Profile	<i>Accept-Profile</i> (header info)	geosparql	<i>na</i>		Surround supports Conneg by profile
Start page	<i>page</i>	3	<i>startIndex</i>	5000	Paging support
Number of records	<i>per_page</i>	40	<i>limit</i>	5000	Paging support

From the perspective of the client developers, it would be helpful to have better agreement between the API semantics.

10.8. Client Enhancements and Future Work

An immediate improvement that the **D138** team continues to work on is the creation of a configuration script by which to launch a `Pyddgin.py` viewer. Such a script would include:

1. URIs of data to load into the viewer
2. Parameters to apply to each layer
3. Basic styling instructions (RGB color and transparency)
 - a. Styling instructions based on attributes.
 - b. (Current styling instructions are hard coded in the application.)

Future requirements to support enhance client viewer functionality include:

1. A method to convey DGGS RS information with the Zone IDs
2. Support for DGGS Data Type as API attribute
3. An API method to specify feature attribute to return
4. CSS based styling support

5. Advanced styling options – (see Cairo reference)
 - a. Outlines - Dashing, Thickness, Smoothing
 - b. Crop fill to outline
 - c. Hashing, Gradients, Image fill support
 - d. Compound styles

Chapter 11. DGGS Enabled Data Services

D139 DGGS Enabled Data Services - DGGS-enabled data services, i.e. either OGC API endpoints or OGC W*S services that understand ZoneIDs as spatial filters. The services can use the DGGS reference implementation library from D137 to convert ZoneIDs to geographic filters, or implement the same DGGS to ensure consistent ZoneIDs across D137, D138, and D139. Alternatively, the instances can make use of the D137 service instance for ZoneID(s) to geographic location conversion.

11.1. OGC API - Features instances

Three instances of the OGC LD API were implemented:

1. **TB16Pix Ref** - TB16Pix Reference Dataset

- Delivers the Zones and Cells of the TB16Pix dataset, i.e. each of its Grids from Grid 0 to Grid 15.
- This dataset contains no data other than the TB16Pix reference grids.

2. **Geofabric** - Australian Hydrological Geospatial Fabric

- A dataset containing a single Collection of hydrological catchments for Australia covering the area of the Australian Capital Territory (ACT) only.
- ~70 features

3. **ASGS** - Australian Statistical Geographies Standard

- A dataset containing a single Collection of Australian census *Statistical Area Level 1* (SA1) blocks covering the area of the Australian Capital Territory (ACT) only.
- ~1,000 features

11.1.1. TB16Pix Ref

Content served by this API instance is mostly generated 'on the fly' by the *rHEALPix Python package* [6]. The API has a static list of DGGS *Grid* instances from refinement level 0 to 15 (of a potentially infinite set of grids, getting ever more fine). Each *Grid* is presented as an OGC API *Collection* instance.

The persistent identifier of <https://w3id.org/dggs/tb16pix-api> was allocated to the API. Therefore the API landing page is at <https://w3id.org/dggs/tb16pix-api>. The URI redirects to a current system implementation of <https://tb16pix.dggs.org>, but this is not guaranteed to persist.

URIs for this dataset for the Dataset (landing page), API description, Collections, a Collection, a Collections Items and a Item (a Feature) are given in the table below. Also given are the Zone's TB16Pix Geometry (trivially a textual representation of the Zone ID) in a pseudo *Well-Known Text* (WKT) [7] geometry representation format.

Table 45. TB16Pix API Endpoint summary

URI	Description
-----	-------------

https://w3id.org/dggs/tb16pix-api	OGC API Landing Page. Also delivers dataset descriptions by ConnegP
https://w3id.org/dggs/tb16pix-api/spec	OGC API specification
https://w3id.org/dggs/tb16pix-api/doc	Open API Documentation for this OGC LD API
https://w3id.org/dggs/tb16pix-api/conformance	Conformance Classes conformed to by this API
https://w3id.org/dggs/tb16pix-api/collections	Collections - the TB16Pix <i>Grid</i> instances, 0 - 15
https://w3id.org/dggs/tb16pix-api/collections/g2	The <i>Grid 2</i> Collection
https://w3id.org/dggs/tb16pix-api/collections/g2/items	Features (Zones) within the <i>Grid 2</i> Collection
https://w3id.org/dggs/tb16pix-api/collections/g2/items/N03	Zone N03

For each **Collection**, such as TB16Pix *Grid*, the list of contained **Feature** instances, such as *Zones* was calculated by *rHEALPixDGGS* [6] and listed on its Features page.

For each **Feature**, (TB16Pix *Zone*, *Parent*, *Neighbour* and *Children* Zones) were calculated by *rHEALPixDGGS* [6] and links to them delivered on a Zone's description page.

11.1.2. Geofabric

Content for this API instance is taken from the *LoCI Project's* (<http://loci.cat/>) implementation of the *Geofabric Surface Network V2.1.1* dataset (<https://data.gov.au/dataset/ds-dga-7bd1ca77-86d3-4e22-bc56-baccadf7bf42>). WGS84 polygonal geometries for *Contracted Catchment* features that cover the area of the ACT were converted to TB16Pix geometries and both geometries are presented by the API, side-by-side, for each feature.

Geometry conversion was done using a modified version of the *rHEALPixDGGS software* [6]. The modifications resulted in a new version of the software.

For content views that support multiple geometries (all views in the API, other than GeoJSON), the geometries are linked to the feature via *GeoSPARQL* [3] ontology constructs.

The persistent identifier <https://w3id.org/dggs/geofabric-api> was allocated to the API, thus its landing page is at <https://w3id.org/dggs/geofabric-api>. The URI redirects to a current system implementation of <https://geofabric.surroundaustralia.com>, but this is not guaranteed to persist.

URIs for this dataset for the Dataset (landing page), API description, Collections, a Collection, a Collections Items and an Item (a Feature) are given in the table below.

Table 46. *Geofabric API Endpoint summary*

URI	Description
https://w3id.org/dggs/geofabric-api	OGC API Landing Page. Also delivers dataset descriptions by ConnegP
https://w3id.org/dggs/geofabric-api/spec	OGC API specification
https://w3id.org/dggs/geofabric-api/doc	Open API Documentation for this OGC LD API

https://w3id.org/dggs/geofabric-api/conformance	Conformance Classes conformed to by this API
https://w3id.org/dggs/geofabric-api/collections	Collections - the single Collection <i>Contracted Catchments (ACT Only)</i> is presently available
https://w3id.org/dggs/geofabric-api/collections/CC	The <i>Contracted Catchments (ACT Only)</i> Collection
https://w3id.org/dggs/geofabric-api/collections/CC/items	Features within the <i>Contracted Catchments (ACT Only)</i> Collection
https://w3id.org/dggs/geofabric-api/collections/CC/items/12104851	Catchment 12104851

11.1.3. ASGS

Content for this API instance is taken from the *LocI Project's* (<http://loci.cat/>) implementation of the *ASGS (2016 Edition) - Boundaries* dataset (<https://data.gov.au/dataset/ds-dga-32adc1ef-5bac-4eaa-9521-a116792f32a1>). WGS84 polygonal geometries for *Statistical Area Level 1 (SA1)* features that cover the area of the ACT were converted to TB16Pix geometries and both geometries are presented by the API, side-by-side, for each feature.

Geometry conversion was done, as per the Geofabric data, using *rHEALPixDGGS software* [6]. Feature/Geometry links were made, as per the Geofabric data, via *GeoSPARQL* [3] ontology constructs.

The persistent identifier <https://w3id.org/dggs/geofabric-api> was allocated to the API, thus its landing page is at <https://w3id.org/dggs/geofabric-api>. The URI redirects to a current system implementation of <https://geofabric.surroundaustralia.com>, but this is not guaranteed to persist.

URIs for this dataset for the Dataset (landing page), API description, Collections, a Collection, a Collections Items and an Item (a Feature) are given in the table below.

Table 47. ASGS API Endpoint summary

URI	Description
https://w3id.org/dggs/asgs-api	OGC API Landing Page. Also delivers dataset descriptions by ConnegP
https://w3id.org/dggs/asgs-api/spec	OGC API specification
https://w3id.org/dggs/asgs-api/doc	Open API Documentation for this OGC LD API
https://w3id.org/dggs/asgs-api/conformance	Conformance Classes conformed to by this API
https://w3id.org/dggs/asgs-api/collections	Collections - the single Collection <i>SA1s</i> is presently available
https://w3id.org/dggs/asgs-api/collections/SA1s	The <i>SA1s</i> Collection
https://w3id.org/dggs/asgs-api/collections/SA1s/items	Features (<i>SA1s</i>) within the <i>SA1s</i> Collection
https://w3id.org/dggs/asgs-api/collections/SA1s/items/80101100101	SA 80101100101

11.2. OGC API - Features architecture

The following software / service architecture was used to implement the three API instances described above.

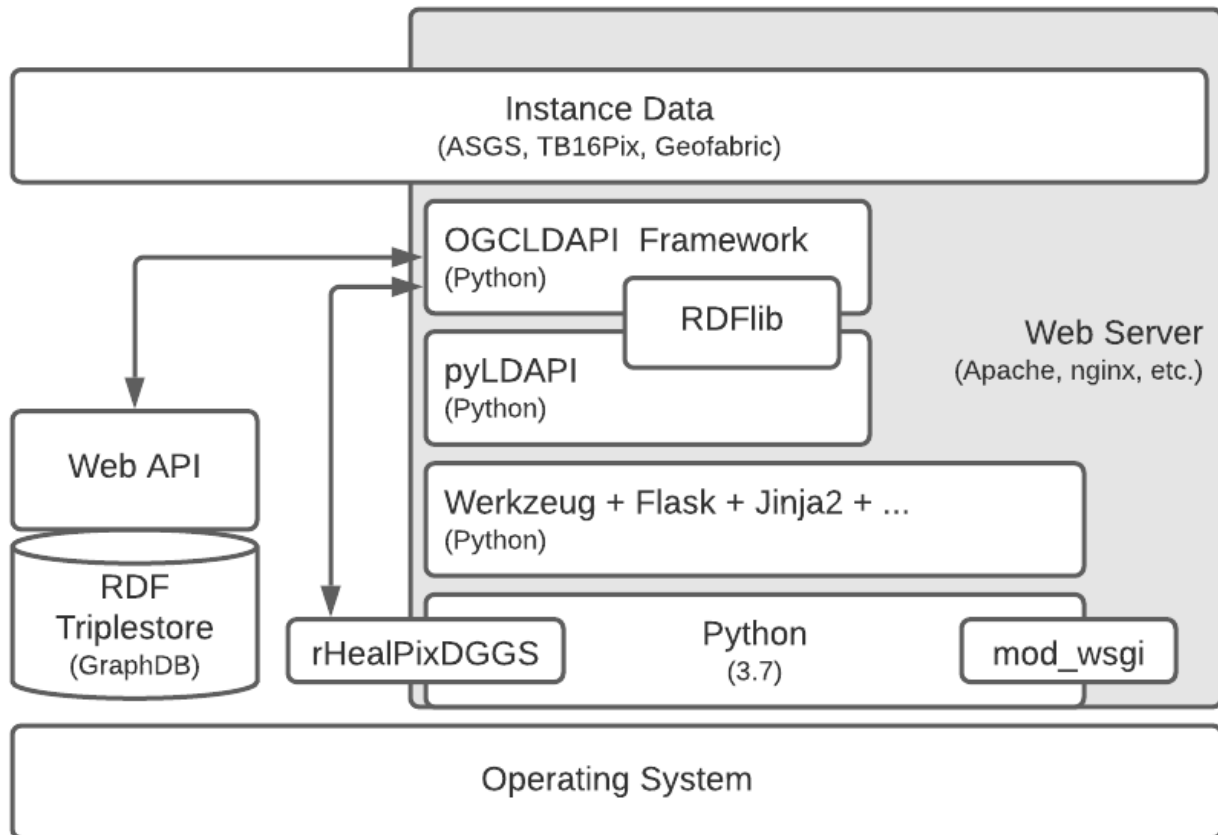


Figure 36. OGC API - Features architecture overview

Progressing through the software stack in the figure above from bottom to top:

- **Operating System:** The servers used to implement these APIs use Linux (Ubuntu 20.04) as their *Operating System* however this stack will work with negligible change on Windows, Mac and other Linux version *Operating Systems*. MacOS has also been used in testing.
- **Web Server:** All requests to the APIs are initially received by a standard HTTP Server application, such as Apache or nginx. *Apache2* is the particular HTTP Server used in all 3 instances.
- **mod_wsgi^[17]:** forwards most HTTP requests to a Python environment via Apache's *mod_wsgi* module which is Apache's implementation of the WSGI protocol.
- **Werkzeug^[18]:** handles HTTP messages passed to it via *mod_wsgi* and, in turn, passes messages on to Python's *Flask* "web framework". This package is available freely via the Python Package Index (PyPI) but is usually installed automatically when *Flask* is installed.
- **Flask^[19]:** provides the definitions of API endpoints and links requests to particular resources, for instance it links most requests for HTML responses to *Jinja2* templates. *Jinja2* is another Python package. This package is available freely via PyPI.

- **Flask additions:** *Flask*, as implemented here, also uses a number of small plugins for HTTP security and other standard *Flask* operations. All these *Flask* "plugin" packages are available freely via PyPI.
- **pyLDAPI^[20]:** rests on top of *Flask* and is a python package supplying a framework within *Flask*'s framework for the handling of Linked Data-style HTTP requests. Specifically here this means the handling of HTTP requests in accordance with *Content Negotiation by Profile* where different HTTP Media Types may be specified and data requested according to different *profiles* (*specifications*). This package is available freely via PyPI.
- **OGCLDAPI Framework^[21]:** is the the final framework in use. It was developed partly for this Testbed. It is an instance of *pyLDAPI* that provides specific endpoints and return formats and content in accordance with the OGC API Features specification. This package is not yet available via PyPI but may become so, after further refactoring. Currently it is available via GitHub as a Git Repository.
- **RDFlib^[22]:** Both *pyLDAPI* and *OGCLDAPI Framework* are heavily dependent on the Resource Description Framework (RDF) manipulation Python package *RDFlib* which is also available freely on PyPI.
- **Instance Data:** The *OGCLDAPI Framework*, via *pyLDAPI* & *RDFlib* accesses RDF data made available to it via *RDFlib*'s *Graph* class. In two of the three instances here, ASGS & Geofabric, the RDF content is stored in an RDF database - a triplestore - and accessed via an API. Both ASGS & Geofabric use the *GraphDB* triplestore and access data via SPARQL queries posed to its API through *RDFlib*'s *Graph* class's *Store* interface. The third instance, TB16Pix Ref, doesn't use a triplestore for data storage as it generates responses to queries on the fly, using the *rHEALPixDGGS* Python package. The responses from *rHEALPixDGGS* are often encoded in RDF.
- **rHEALPixDGGS^[23]:** is a Python Package freely available on PyPI and calculates relations between *rHEALPix* (and this *TB16Pix*) elements. *rHEALPixDGGS* is dependent on a number of standard scientific Python packages such as *_numpy_*, all of which are available for free on PyPI.
- **GraphDB^[24]:** the RDF triplestore and its API are available from its vendor's website. The free edition is used for these instances.

The Python dependencies of the API are summarized and machine-installable within the *OGCLDAPI Framework*'s *requirements.txt* file which can be used in conjunction with PyPI. This file is found, with installation instructions, within the *OGCLDAPI Framework*'s version control repository (see link above).

11.3. Supporting Assets

To support these deployments of the OGC API - Features instances accessing DGGS content, a number of semantic assets were generated that describe the data models used within the APIs' data sources. This was required for the APIs also and defined, as supersets of the core OGC API - Features data model. These assets, as well as the API Framework, test client and API instances, were listed in a temporary DGGS catalogue, online at <https://w3id.org/dggs/cat>.

11.3.1. Supporting software tools

Within the DGGS catalogue, the following non-API instances of software assets relevant to these API

instances' deployment are listed:

Table 48. DGGs Software Assets

URI	Title	Description	Role
https://w3id.org/dggs/cat	DGGS Catalogue	A DCAT2-compatible Dataset containing a catalogue of things, such as services, dataset, models, vocabularies, relating to Discrete Global Grid systems	This lists all the DGGS software, API instances and semantic assets described in this chapter of this report
https://w3id.org/dggs/rhealpixdggs	rHEALPix Discrete Global Grid System software	rearranged HEALPix DGGS - the rHEALPix DGGS software library	This software is used to calculate the TB16Pix Zone IDs within the TB16Pix dataset for delivery by the <i>TB16Pix Ref</i> API
https://w3id.org/dggs/dggs/v	DGGS Geometry Validator	A Python library for validating multiple DGGS geometry literal values	To validate DGGS literals independent from any other API or implementation system that needs to produce or consume them

11.3.2. Semantic Assets

Within the DGGS catalogue, the following semantic assets relevant to these API instances' deployment are listed

Table 49. DGGs Semantic Assets

URI	Title	Description	Role	Ref
https://w3id.org/dggs/as	DGGS Abstract Specification Ontology	An ontology describing the parts of a Discrete Global Grid System in Semantic Web terms, derived from the OGC's DGGS Abstract Specification	This ontology is needed to allow the OGC LD APIs to deliver Semantic Web (RDF) data and also to map between DGGS Abstract Specification elements and API elements	[8]

https://w3id.org/dggs/as-terms	DGGS Abstract Specification 2.0 Terms and definitions vocabulary	Terms, represented as Simple Knowledge Organization System (SKOS) Concepts, from Section 4 of the Discrete Global Grid Abstract Specification, version 2.0	Presents the DGGS Abstract Specification terms in machine-readable format so they can be accessed within Semantic Web applications. The OGC API's definitions for objects such as Feature are linked to these definitions	[9]
-----------------------------------------------------------------------------	------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----

The following semantic assets were not created for these API deployments but are relied on by them:

Table 50. Semantic Assets Depended Upon

URI	Title	Description	Role	Ref
http://www.opengis.net/doc/IS/geosparql/1.0	OGC GeoSPARQL - A Geographic Query Language for RDF Data	An RDF/OWL vocabulary for representing spatial information	The ontology used for the most basic (fundamental) Semantic Web representations of Feature and Geometry objects. The DGGS element-to-GeoSPARQL mapping is contained in the <i>DGGS Abstract Specification Ontology</i> (see above)	[3]

https://linked.data.gov.au/def/geox	GeoSPARQL Extensions Ontology	An extension to GeoSPARQL with new features for the representation of additional elements of feature geometry, such as spatial-resolution, length, area and volume.	Needed for properties such as <i>asDGG</i> which link a (Semantic Web) representation of a <i>Geometry</i> to its literal representation. GeoSPARQL on its own doesn't contain all the properties needed for this. This ontology has been extended with TB16 DGGs work in mind and informs GeoSPARQL 1.1 revision	[4]
http://www.w3.org/ns/dx/conneg/alt	Alternative Profiles Ontology	This ontology allows for the description of representations of Internet resources	Use by the OGC LD API implementations to communicate their various <i>profile</i> views of objects. See, for any OGC API instance above: <code>{Feature URI}?_profile=alt</code>	[10]

11.3.2.1. DGGs Abstract Specification Ontology

The DGGs Abstract Specification Ontology was created to provide a bridge between DGGs and systems such as GeoSPARQL [3] for which it is necessary to have Semantic Web definitions of elements.

Figure 35, taken from the ontology's HTML documentation, gives an overview of the ontology's classes and properties.

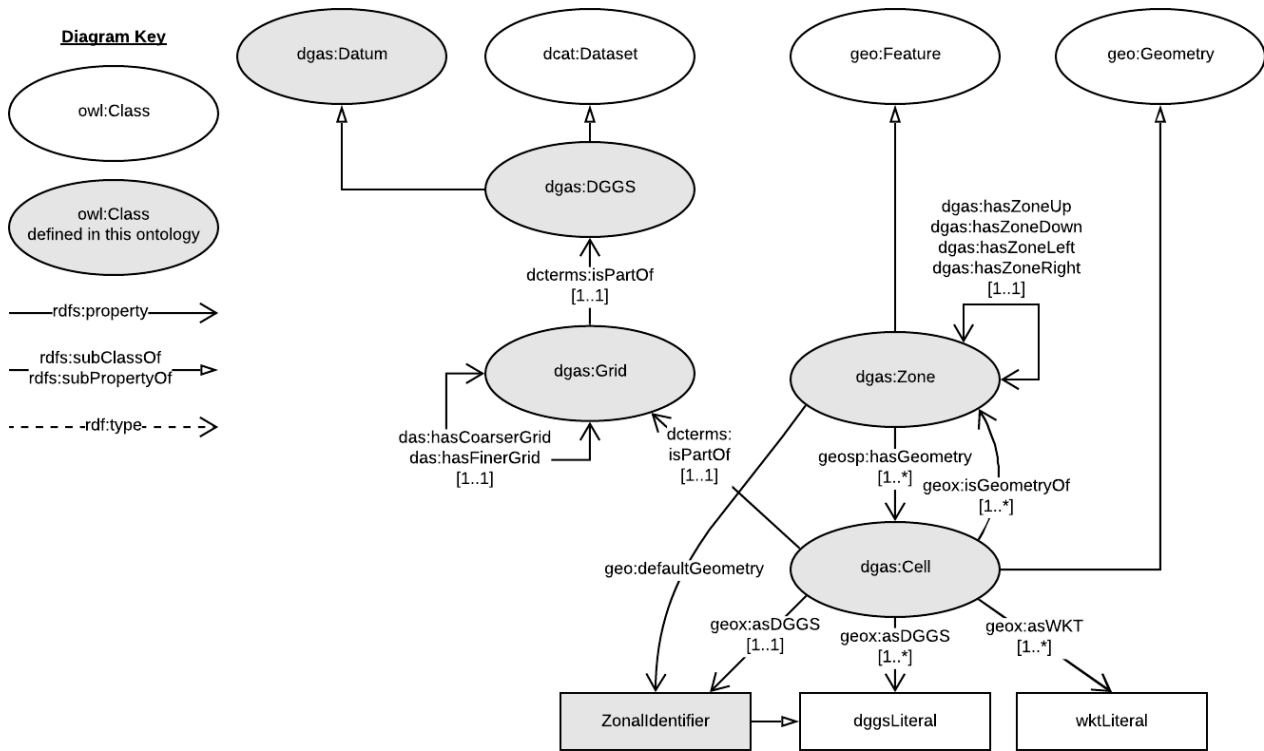


Figure 37. DGGs Abstract Specification overview

The Ontology is a *profile* of GeoSPARQL and interprets DGGs **Grid**, **Zone** & **Cell** concepts within the GeoSPARQL **Feature/Geometry** framework: a **Zone** is a specialized **Feature** etc. Nothing in GeosPARQL corresponds to the notion of a feature collection so there is no GeoSPARQL generic version of a DGGs **Grid**.

The modelling experience in this Testbed was straightforward: major DGGs elements can be housed fairly uncontroversially within GeoSPARQL, however refinement of this ontology will be needed as GeoSPARQL itself undergoes updates (the GeoSPARQL Standards Working Group is working on a next version coincidentally with this Testbed but GeoSPARQL 1.1 is expected only approximately 6 months after the Testbed concludes) and as the DGGs AS 2.0 is published. Additionally, there is known work to better reuse elements of fundamental spatial ontologies within this ontology for generic concepts that have, currently been defined within the ontology, such as **Datum** - a *reference frame that realizes the positions of the origin, the scale, and the orientation of a coordinate system*.

11.3.2.2. DGGs Abstract Specification 2.0 Terms and definitions vocabulary

The DGGs Abstract Specification v2.0 (DGGs AS) vocabulary was derived directly from the terms and definitions provided in the draft Topic 21 DGGs Abstract Specification v2.0 however it is presented in Semantic Web terms, according to the widely used SKOS model for vocabularies ^[25].

Publication in this way allows machine-readable forms of AS element definitions and will assist with OGC APIs for terms delivered in this way can be individually referenced in API data. This means when an API communicates a term, the API needs just link to the vocabulary element rather than storing, and duplicating, the vocabulary's content or linking to the entire vocabulary or specification.

The OGC presents many specifications' terms as SKOS vocabularies via its Naming Authority

however there is currently no automated or even specified workflow to generate such semantic assets from the specification documents.

[17] <https://modwsgi.readthedocs.io>

[18] <https://pypi.org/project/Werkzeug/>

[19] <https://pypi.org/project/Flask/>

[20] <https://pypi.org/project/pyldapi/>

[21] <https://w3id.org/dggs/ogcldapi>

[22] <https://pypi.org/project/rdfliib/>

[23] <https://pypi.org/project/rHEALPixDGGS/>

[24] <https://graphdb.ontotext.com>

[25] <https://www.w3.org/TR/skos-reference/>

Chapter 12. Future Tasks

Testbed-16 provided an opportunity to help highlight and bring into focus specific activities and actions that need to be undertaken to support the increased and widespread implementation and adoption of standardized DGGs technologies across the geospatial sector. These include (but are not limited to) the following:

1. Maturing DGGs Reference Library Implementations to bring them into conformance with OGC Topic 21 v2.0;
2. Drafting and elaboration of OGC APIs for DGGs;
3. Exploring the opportunities and limitations of DGGs driven analytics;
4. Implementation of OGC Registries for DGGs Implementations and DGGs enabled data services;
5. Targeted DGGs Interoperability Experiments.

12.1. Maturing DGGs Reference libraries to meet community & future testbed needs

Table 1 provides a list of open-source and proprietary reference DGGs libraries. Because these DGGs libraries were developed prior to the publication of OGC Abstract Specification Topic 21 v2.0 (ISO/DIS 19170-1) there are likely to be inconsistencies between these DGGs libraries and the conformance classes of the soon to be published version of the Core DGGs Standard. This provides a driver for DGGs developers to improve these reference libraries to bring them into alignment with the new conformance classes described by the new DGGs Standard.

There is an opportunity through the OGC Innovation Program (through Testbeds, Pilots and Interoperability Experiments) to support this work and to enable further improvements to be made to the OGC/ISO DGGs Standards suite based on the experience of implementing these standards.

Based on the outcomes from this Testbed activity the following DGGs implementation development tasks have been identified as potential tasks for inclusion in future OGC Innovation Program activities:

12.1.1. Development tasks identified for H3

1. H3 does not enforce precise equal area constraints on zones across the globe, nor does it currently support higher dimensional DGGs Reference System specifications. This could be improved by the following development tasks:
 - a. Inclusion of a method/function to enforce equal area zones to a prescribed level of precision;
 - b. Extension of H3 to support 3D (volumetric) and 2D/3D+T (spatio-temporal) DGGs Reference System specifications.
2. An additional, task worthy of consideration (although not critical for H3 to demonstrate conformance with OGC Topic 21) is the extension of the DGGs hierarchy to include additional refinement ratios other than 1:7.

12.1.2. Development tasks identified for rHEALPix

1. While rHEALPix is perhaps one of the 'most' conformant reference DGGs libraries in relation to OGC Topic 21 v2.0, there are some specific development tasks that could be undertaken to extend and improve this library to enable it to be better suited to the requirements of current and emerging needs.
 - a. Support for multi-threaded operation to support concurrency scaling, probably through a C implementation.
 - b. A native `javascript` implementation to support front ends that integrate traditional and DGGs data and cannot just rely on zone identifiers.
 - c. Extension to 3D and 2D/3D +T DGGs Reference Systems.
 - d. Examples and tutorials to assist developers in uptake.

12.1.3. Development tasks identified for both H3 and rHEALPix

Separately to the identified areas for individual enhancement of the H3 and rHEALPix DGGs libraries there are a number of actions identified that will improve the implementation of both DGGs libraries. These include:

1. The addition of a set of functions that fully align with OGC Topic 21 v2.0 ZoneQuery;
2. Addressing the performance issues identified during Testbed-16;
 - a. There was an enormous amount of triplicate, quadruplicate, and worse, processing done in the way that boundary calculations were requested within GeoServer;
3. Identifying functions (other than ZoneQuery) that are candidates for standardization. Such as:
 - a. A standardized way of defining DGGs RS, e.g. WKT;
 - b. A standardized way of extracting zone edges as sequences of vertices or centroids of higher resolution zones; and,
 - c. Data quantization functions from point, line, polygon, point cloud, raster that explicitly support quantization roles.

12.1.4. Development tasks relevant to other DGGs libraries

Additional development tasks that are of a more general nature include (inter alia):

1. Identifying and cataloging other existing or emerging DGGs libraries that are candidates for being DGGs reference libraries.
2. Comparison and contrast of DGGs library characteristics;
3. Determining the effort required to implement DGGs libraries and make them available for wider use. This recognizes that, many open-source software tools require a significant effort to integrate them seamlessly into production Spatial Data Infrastructures, and this is even more visible with DGGs libraries because of the "Big Data" use case scenarios they are suited to. Organizations seeking to implement DGGs technologies need to be able to clearly understand the true costs involved with the implementation of these infrastructures.

12.2. OGC API(s) for DGGs

The two OGC APIs that were implemented under the DGGs thread in Testbed-16 have some common characteristics in terms of mapping DGGs on to existing OGC API patterns. There are a number of questions this work has raised that require further exploration through both the OGC Standards and OGC Innovation Programs. These include:

1. How widely applicable are the DGGs-centric API patterns to other OGC APIs?
2. Does the simplicity, and generality of the DGGs spatio-temporal data model lead to an overarching simplicity of OGC API patterns?
 - a. Can a few spatio-temporal OGC API patterns for DGGs replace the needs of multiple existing OGC APIs?
 - b. Is the proposal for a WKT representation for DGGs geometries introduced in [Proposal for OGC WKT for DGGs geometries](#) sufficient?
 - c. If all the geometries for vector, raster, point-cloud, tiles, maps and social media can be reduced to small set of geometries, such as those in [Proposal for OGC WKT for DGGs geometries](#), can a single DGGs variant of OGC API features be used for all feature types?
 - d. How far can we extend a single set of standardized DGGs processes that can process any mixture of these geometry types? This could result in a DGGs variant of OGC API - Processes that included a standardized set of DGGs Processes. These processes could augment the DGGs Core operations in ZoneQuery to provide both a wider suite of domain agnostic spatial analytics, and a standardized way of partitioning jobs based on DGGs tiling.
3. DGGs have an explicit discretized spatio-temporal resolution and precision that aligns with the zone hierarchy, and is exposed in the OGC API - Features and Processes as a structure in the collections. Giving the user specific and consistent access to precision is therefore an opportunity that needs further discussion. This is elaborated further in [\[DGGs_processing_opportunities\]](#) below.

While the differences between a DGGs-centric (or DGGs enabled) OGC API implementation and conventional OGC API - Features or Processes implementations are subtle, they are quite distinct. This distinction, and the opportunity to derive additional spatial data integration capabilities through DGGs implementations via OGC API mechanisms support the concept of drafting an OGC API DGGs specification.

This activity should be conducted with close coordination between the DGGs SWG and the OGC Innovation Program activities in a similar fashion to the other OGC API initiatives that are currently under way.

12.3. DGGs processing opportunities

12.3.1. Pre-built multi-resolution statistics

One of the data processing and analytics considerations discussed by participants of Testbed-16 DGGs thread was the concept of statistical pyramiding of data observations. This involves storing/mapping the raw data values to DGGs zones at the finest resolution (ZoneLevel) relevant to

the data (and its level of precision) and then computing statistical aggregations of those values for each successive parent Zone until the base DGGs resolution (i.e. ZoneLevel = 0) is reached. DGGs enabling data in this way provides mechanisms for rapid threshold query filtering of data at lower resolutions to identify and zoom in on areas of interest in a particular dataset without having to perform multiple high resolution queries of the entire dataset.

Storing full statistics at every resolution is probably overkill, but storing full statistics at every third or fourth level could be highly advantageous without costing a great deal of storage. What constitutes 'full statistics' is probably dependent on the attribute type. For classified data a histogram of frequency of each value might be sufficient. For continuous surface data the population distribution (eg Poisson vs normal), mean, standard deviation, min and max might be appropriate and so forth.

As with all Big Data scenarios, there are some trade-offs for DGGs enabled data custodians/providers to make in balancing the cost of additional storage that multi-resolution pyramiding will require against the application and analysis efficiencies gained by the generation of DGGs enabled lookup tables.

For time critical applications, such as disaster response data integration, the operational benefits justify the additional storage costs. For other, less time dependant applications, additional processing time will be an acceptable trade-off in order to minimise data management overheads. The key thing to note here is that DGGs infrastructures provide the flexibility to traverse between these two end member data management strategies as operational requirements demand.

This concept could be explored further as part of the OGC DGGs API work; particularly considering the implementation and standardisation of a DGGs analogue of OGC Tiles API. This analogue DGGs tiles approach could consider the OGC API implementation of ZoneTags, DataTiles and GraphicTiles DGGs quantization strategies defined in OGC Topic 21 - Part 1 v2.0.

12.3.2. Just in time precision

A key benefit of DGGs infrastructures is that, unlike conventional GIS infrastructures, there is a direct and finite precision associated with the DGGs structure (as apposed to a reference to data precision in metadata). This enables the zones of a DGGs to be used directly to represent the precision of each observation - even in datasets that contain variable precision data.

The concept of "Just in Time Precision" is related to the ability to drill down through the DGGs zone hierarchy until the appropriate DGGs resolution (ZoneLevel) is achieved. The principles of statistical pyramiding of data described in the previous subsection can be applied to the intermediate DGGs levels; enabling targeted spatial filtering to be achieved based on Zone IDs and statistical summaries.

This is quite different from traditional approaches to risk, uncertainty, and precision. Traditionally the data is processed at a predetermined spatial resolution and a result is determined. Then an additional analysis is done to determine the precision or error bars or statistics associated the result, and a choice is then made as to caveats that should be given to the result given the purpose of the analysis.

With the appropriate data architecting this can be achieved on the fly as data is streamed into the

DGGS infrastructure. Allowing a flexible association of data at multiple resolutions in a "Just-in-time" fashion.

DGGS provides the opportunity to include the desired precision of the result in the API.

The processing would start at a coarse spatio-temporal resolution near the top of the hierarchy, and then traverse down those parts of the hierarchy that need to be processed to determine a result and stop either when the resolution of the data is reached or when the result is sufficiently precise for the purpose.

This concept could be explored further as part of the OGC DGGS API work; particularly considering the implementation and standardisation of a DGGS analogue of OGC Tiles API. This analogue DGGS tiles approach could consider the OGC API implementation of ZoneTags, DataTiles and GraphicTiles DGGS quantization strategies defined in OGC Topic 21 - Part 1 v2.0.

Examples of potential use cases include:

- Decision making in disaster response.
- Triaging objects for processing in autonomous navigation.
- Managing a pandemic using a risk based approach, as distinct from a rule based approach.

12.4. DGGS Analytics - What Does that Really Mean?

A well recognised issue with the DGGS standards baseline is that, to date, there have been limited examples of DGGS infrastructures in action demonstrating data analytics applications. Some notable exceptions to this have been the PYXIS/Global Grid Systems Inc. and the Uber/Unfolded Inc. DGGS infrastructures.

With the growing maturity of both the DGGS standards baseline and DGGS implementations currently being developed by multiple organizations around the world, along with the emergence of OGC APIs, there is a great opportunity to explore, experiment and standardise a set of common DGGS analytic functions that can support data integration into and across multiple DGGS (and non-DGGS) Infrastructures.

A key success criteria for this; however, will be agreement on what the term DGGS Analytics really means. This activity should be undertaken as a joint activity between the DGGS DWG/SWG and OGC Innovations Program - facilitating the strategic standards related discussions and codification of this topic based on the guidance from active implementation and testing made possible through the OGC Innovations Program.

As DGGS infrastructures become more widely adopted it will be increasingly important for the standardisation of common analytical functions. Targeted use case scenario development with DGGS technologies in mind through upcoming OGC Testbeds, Pilots and Interoperability Experiments will be important.

12.5. The evolution of the OGC DGGS Registry

The concept of an OGC DGGS Registry has evolved somewhat during the course of this Testbed

activity. From merely describing the various profiles of DGGs implementation specifications to also include DGGs enabled data sources.

With the increasing demand for DGGs implementations by organizations across the entire spatial sector, the importance of an official registry of DGGs profiles and DGGs enabled datasets has been recognized. The participants recommend that the OGC strongly consider the funding of this work through the OGC Innovation Program activities. Research & development questions that should be elaborated include:

1. The enhancement of the existing prototype OGC DGGs profile Registry to include conformance testing against OGC Topic 21 v2.0;
2. The discovery and cataloguing of DGGs enabled data sources;
3. The development of an OGC Registry of DGGs enabled data sources;
4. The integration of the OGC Registry of DGGs profiles and the OGC Registry of DGGs enabled data sources.

12.6. Opportunities for DGGs API in Interoperability Experiments

Much of the effort moving forward will be exploring and developing standardized mechanisms to support DGGs to DGGs interoperability will be conducted in the context of OGC DGGs APIs. Coordination between the OGC API Features, Processes and Common future activities, along with OGC API DGGs development activities, will be important.

The following DGGs Features API extensions/demonstrations were discussed but not implemented during Testbed 16:

1. Deliver of gridded data

- a. It would be possible to create APIs for a dataset that contains an OGC API *Collection* per DGGs *Grid*, each containing an OGC API *Feature* for each DGGs *Zone* instances, as per the TB16Pix Ref but with additional data values per *Zone*.
- b. This would allow the delivery of gridded data via the Features API which means OGC API filtering and so on could be performed with response payloads containing gridded data in a datum-per-Cell format, yet to be determined.

2. Implement multiple DGGs geometry values, side-by-side

- a. To compare directly the use of multiple DGGs, a Features API could deliver multiple Geometries, encoded according to different DGGs, for Features.
- b. This is akin to the current WGS84/TB16Pix side-by-side geometries but, if this was enabled, BBOX and similar API features would need to include DGGs markers within them or else the filter value would be ambiguous since multiple DGGs may share the same *Zone ID*, for instance both TB16Pix and AusPix contain a *Zone ID* of P1234.

3. Improve back-end native DB DGGs functions

- a. The current DGGs OGC API - Features instances use RDF triplestores for their data storage and take advantage of GeoSPARQL functions for non-DGGs spatial functions, such as WGS84

Bounding Box filters. No triplestores yet implement GeoSPARQL functions for DGGs data and thus the DGGs Bounding Box filters have required implementation in application code (Python) and this is inefficient and slow.

- b. Future DGGs OGC API - Features would benefit from triplestores - or any other back-end data store - that can natively process GeoSPARQL features using DGGs geometries, such as `geof:sfContains`, which could then be used to answer a DGGs Bounding Box filter query.
- c. Due to the simplicity spatial query when using DGGs geometries, custom but simple SPARQL functions could be written to emulate `geof:sfContains` and need not implement the lower-level functionality required by GeoSPARQL implementations which use tools such as GDAL under-the-hood.

4. Choose better geometry types

- a. Geometry types for DGGs data echoing regular, non-DGGs, geometry types where implemented. It is not known whether the use of regular-style geometries will allow for all DGGs possibilities as opposed to DGGs *native* geometry types.
- b. *Native* geometry types are not fully codified and tested so this needs to occur before assessments of what geometry types can be used effectively for DGGs data into the future.
- c. The DGGs Geometry Validator software.^[26] has been established to validate DGGs geometry literal values and this will help with their testing.

5. Implement non-square BBOX filters

- a. Currently only Cell ID-based bounding box filters for features have been demonstrated in the API instances. Filters using a pair of Cell IDs to allow for non-square and square but non-Cell border-aligned square bounding box filtering should be implemented as the very next OGC API: Features DGGs feature.
- b. Non-quadrilateral filtering would be yet a further API capability to add.

[26] <https://github.com/surroundaustralia/dggsgv>

Appendix A: Revision History

Table 51. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
May 25, 2020	B Cochrane, R Gibb	.1	all	initial version
May 31, 2020	B Cochrane, R Gibb, M Purss	.2	all	Finalising IER
Oct 19, 2020	A Aime, R Gibb	.3	7.8	Draft OGC API - Processes definition
Oct 19, 2020	A Aime	.3	9	Draft D137 implementation
Oct 19, 2020	B Cochrane	.3	10	Draft D138 implementation
Oct 19, 2020	B Cochrane, R Gibb, M Purss	.3	all	Finalising DER
Oct 20, 2020	M Purss, R Gibb	.3	12	Draft Future Tasks
Oct 21, 2020	B Cochrane	.3	10	Draft D137 implementation
Oct 22, 2020	N Car, R. Gibb	.3	7.7	Draft OGC API - Features definition

Appendix B: Bibliography

- [1] Svensson, L.G., Atkinson, R., Car, N.J.: Content Negotiation by Profile. W3C Dataset Exchange Working Group (2018).
- [2] Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax. W3C (2014).
- [3] OGC GeoSPARQL - A Geographic Query Language for RDF Data. Open Geospatial Consortium (2012).
- [4] Car, N.J., Cox, S.J.D.: GeoSPARQL Extensions Ontology, <https://linked.data.gov.au/def/geox>, (2019).
- [5] Vretanos, P.(P.A.: OGC Testbed-16: Data Access and Processing Engineering Report. Open Geospatial Consortium, <http://docs.opengeospatial.org/per/20-16.html> (2021).
- [6] Raichev, A., Gibb, R.: rHEALPixDGGS, <https://pypi.org/project/rHEALPixDGGS/>, (2020).
- [7] ISO/IEC 13249-3:2016: Information technology – Database languages – SQL multimedia and application packages – Part 3: Spatial. International Organization for Standardization (2016).
- [8] Car, N.J.: DGGS Abstract Specification Ontology, <https://w3id.org/dggs/as>, (2020).
- [9] Car, N.J.: DGGS Abstract Specification 2.0 Terms and definitions vocabulary, <https://w3id.org/dggs/as-terms>, (2020).
- [10] W3C Dataset Exchange Working Group: Alternative Profiles Ontology, <http://www.w3.org/ns/dx/conneg/altr>, (2019).