OGC API - Maps - Part 1

Core

Open Geospatial Consortium

Submission Date: 2024-02-15

Approval Date: <yyyy-mm-dd>

Publication Date: 2024-05-01

External identifier of this OGC® document: https://www.opengis.net/doc/IS/ogcapi-maps-1/1.0

Internal reference number of this OGC® document: 20-058

Version: 1.0.0rc2

Category: OGC® Implementation Specification

Editors: Joan Masó and Jérôme Jacovella-St-Louis

OGC API - Maps - Part 1: Core

Document type: OGC®ImplementationSpecification

Document subtype: if applicable

Document stage: Draft

Document language: English

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.

Table of Contents

1. Scope . . . 13
2. Conformance . . . 14
   2.1. Requirements classes defining resources . . . 14
   2.2. Requirements classes defining parameters . . . 15
   2.3. Requirements classes defining origins . . . 18
   2.4. Requirements classes defining representations . . . 19
   2.5. Summary of conformance URIs . . . 20
3. References . . . 22
4. Terms and Definitions . . . 23
   4.1. **coordinate reference system** . . . 23
   4.2. **coordinate system** . . . 23
   4.3. **data cube (or datacube)** . . . 23
   4.4. **geographic information** . . . 23
   4.5. **height** . . . 23
   4.6. **map** . . . 24
   4.7. **map background** . . . 24
   4.8. **portrayal** . . . 24
   4.9. **rendering device** . . . 24
   4.10. **scale** . . . 24
   4.11. **viewport** . . . 24
   4.12. **width** . . . 24
   4.13. **Web API** . . . 25
5. Conventions . . . 26
   5.1. Identifiers . . . 26
   5.2. Link relations . . . 26
   5.3. Use of HTTPS . . . 27
6. Overview . . . 28
   6.1. Introduction . . . 28
   6.2. Map interoperability . . . 29
   6.3. How to approach an OGC API . . . 30
   6.4. *OGC API - Maps* within the OGC API family . . . 32
      6.4.1. What is a map? . . . 32
      6.4.2. Implementing *OGC API - Maps* within a Web API . . . 32
      6.4.3. Dynamic and scalable map viewers . . . 33
      6.4.4. Client-side maps versus server-side maps . . . 33
   6.5. Description of the domain . . . 34
   6.6. Subsetting and scaling the map . . . 34
   6.7. Available formats and map response expectations . . . 37

7. Requirement Class "Core" . . . 38
7.1. Overview . . . 38
7.2. Map resource . . . 39
7.2.1. Operation . . . 39
7.2.2. Response . . . 39
7.2.3. Recommendations . . . 41
7.3. Declaration of conformance classes . . . 42
8. Requirement Class "Map Tilesets" . . . 44
8.1. Overview . . . 44
8.2. Link from data resource . . . 44
8.3. Map Tileset . . . 45
8.3.1. Operation . . . 45
8.3.2. Response . . . 46
9. Requirement Class "Background" . . . 47
9.1. Overview . . . 47
9.2. Map operation . . . 47
9.2.1. Parameters `transparent` and `bgcolor` . . . 47
9.2.2. Response . . . 54
10. Requirement Class "Collection Selection" . . . 56
10.1. Overview . . . 56
10.2. Operation . . . 56
10.2.1. Parameter `collections` . . . 56
10.2.2. Response . . . 57
10.2.3. Error conditions . . . 58
10.3. Service Metadata . . . 58
11. Requirement Class "Scaling" . . . 60
11.1. Overview . . . 60
11.2. Map Operation . . . 60
11.2.1. Parameters `width` and `height` . . . 60
11.2.2. Parameter `scale-denominator` . . . 63
11.2.3. Response . . . 65
11.2.4. Error conditions . . . 65
11.3. Scale and aspect ratio considerations (informative) . . . 66
11.4. Service Metadata . . . 68
12. Requirement Class "Display Resolution" . . . 71
12.1. Overview . . . 71
12.2. Map Operation . . . 71
12.2.1. Parameter `mm-per-pixel` . . . 71
12.2.2. Response . . . 73
12.2.3. Error conditions . . . 73
13. Requirement Class "Spatial subsetting" . . . 74

13.1. Overview . . . 74
13.2. CRS for subsetting . . . 74
13.2.1. Parameter bbox-crs . . . 75
13.2.2. Parameter subset-crs . . . 76
13.2.3. Parameter center-crs . . . 77
13.2.4. CURIE permission . . . 78
13.3. Subsetting coordinates . . . 78
13.3.1. Parameter bbox . . . 78
13.3.2. Parameter subset . . . 80
13.3.3. Parameter center, width and height . . . 83
13.3.4. Scale and aspect ratio considerations when subsetting . . . 84
13.4. Response . . . 84
13.5. Error conditions . . . 85
14. Requirement Class "Date and Time" . . . 86
14.1. Overview . . . 86
14.2. Describing the temporal extent . . . 86
14.3. datetime query parameter request and response . . . 86
14.4. subset=time query parameter request and response . . . 88
14.5. Actual date & time response header . . . 91
14.6. Closest date & time permission . . . 91
14.7. Response . . . 92
14.8. Error conditions . . . 92
15. Requirement Class "General Subsetting" . . . 93
15.1. Overview . . . 93
15.2. Describing the extent of the additional dimensions . . . 93
15.3. subset query parameter . . . 93
15.4. Response . . . 95
15.5. Error conditions . . . 95
16. Requirement Class "Coordinate Reference System" . . . 96
16.1. Overview . . . 96
16.2. Operation . . . 96
16.2.1. Parameter crs . . . 96
16.2.2. Response . . . 97
16.2.3. Error conditions . . . 97
17. Requirement Class "Orientation" . . . 99
17.1. Overview . . . 99
17.2. Operation . . . 99
17.2.1. Parameter orientation . . . 99
17.2.2. Response headers . . . 100
17.2.3. Response . . . 100
17.2.4. Error conditions . . . 100

18. Requirement Class "Custom Projection CRS" . . . 101
18.1. Overview . . . 101
18.2. Operation . . . 101
18.2.1. Parameter `crs-proj-method` . . . 101
18.2.2. Parameter `crs-proj-params` . . . 101
18.2.3. Parameter `crs-proj-center` . . . 102
18.2.4. Parameter `crs-datum` . . . 103
18.2.5. Response headers . . . 103
18.2.6. Response . . . 104
18.2.7. Error conditions . . . 104
18.3. Available projections . . . 104
18.3.1. Available projections operation . . . 104
18.3.2. Available projections response . . . 105
19. Requirement Class "Collection Map" . . . 110
19.1. Overview . . . 110
19.2. General . . . 110
19.3. Geospatial data resources . . . 110
19.4. Geospatial Data Resource Map . . . 113
19.4.1. Map path . . . 114
19.4.2. Response . . . 114
20. Requirement Class "Dataset Map" . . . 115
20.1. Overview . . . 115
20.2. General . . . 115
20.3. Dataset API landing page . . . 115
20.3.1. Response . . . 115
20.4. Dataset maps . . . 117
20.4.1. Map path . . . 117
20.4.2. Response . . . 118
21. Requirement Class "Styled Maps" . . . 119
21.1. Overview . . . 119
21.2. Styled resources . . . 119
21.3. Styled Resource Map . . . 120
21.3.1. Map path . . . 120
21.3.2. Response . . . 120
22. Requirements classes for encodings . . . 121
22.1. Overview . . . 121
22.2. Requirement Class "PNG" . . . 121
22.3. Requirement Class "JPEG" . . . 122
22.4. Requirement Class "JPEG XL" . . . 123
22.5. Requirement Class "TIFF" . . . 124
22.6. Requirement Class "SVG" . . . 125

22.7. Requirement Class "HTML" . . . 126
23. Requirements Class "API Operations" . . . 128
23.1. Overview . . . 128
23.2. Web API description . . . 128
23.2.1. Response . . . 128
24. Requirements Class "CORS" . . . 132
24.1. Overview . . . 132
24.2. CORS for javascript clients . . . 132
Annex A: Conformance Class Abstract Test Suite (Normative) . . . 133
A.1. Conformance Class "Core" . . . 133
A.1.1. Requirement Map Operation . . . 133
A.1.2. Requirement Map Response . . . 133
A.1.3. Requirement Map Conformance Success . . . 134
A.2. Conformance Class "Map Tilesets" . . . 134
A.2.1. Requirement desc-links . . . 134
A.2.2. Requirement tiles-parameters . . . 134
A.3. Conformance Class "Background" . . . 135
A.3.1. Requirement `bgcolor` parameter definition . . . 135
A.3.2. Requirement `transparent` parameter definition . . . 135
A.3.3. Requirement Background Map Success . . . 135
A.4. Conformance Class "Collection Selection" . . . 135
A.4.1. Requirement `collections` parameter definition . . . 135
A.4.2. Requirement Collection Selection Response . . . 136
A.5. Conformance Class "Scaling" . . . 136
A.5.1. Requirement `width` parameter definition . . . 136
A.5.2. Requirement `height` parameter definition . . . 136
A.5.3. Requirement `scale-denominator` parameter definition . . . 137
A.6. Conformance Class "Display Resolution" . . . 137
A.6.1. Requirement `mm-per-pixel` parameter definition . . . 137
A.6.2. Requirement Display Resolution Map Success . . . 137
A.7. Conformance Class "Spatial Subsetting" . . . 138
A.7.1. Requirement `bbox-crs` parameter definition . . . 138
A.7.2. Requirement `subset-crs` parameter definition . . . 138
A.7.3. Requirement `center-crs` parameter definition . . . 138
A.7.4. Requirement `bbox` parameter definition . . . 139
A.7.5. Requirement spatial subsetting `subset` parameter definition . . . 139
A.7.6. Requirement map subset response . . . 139
A.7.7. Requirement `center` parameter definition . . . 139
A.7.8. Requirement subsetting `width` and `height` parameters definition . . . 140
A.7.9. Requirement map subset success . . . 140
A.8. Conformance Class "Date and Time" . . . 140

A.8.1. Requirement `datetime` parameter definition . . . 140
A.8.2. Requirement `datetime` parameter response . . . 141
A.8.3. Requirement temporal `subset` parameter definition . . . 141
A.8.4. Requirement temporal subset response . . . 141
A.8.5. Requirement temporal axis . . . 141
A.8.6. Requirement temporal subsetting success . . . 142
A.9. Conformance Class "General Subsetting" . . . 142
A.9.1. Requirement uniform additional dimensions . . . 142
A.9.2. Requirement general subsetting `subset` parameter . . . 142
A.10. Conformance Class "Coordinate Reference System" . . . 143
A.10.1. Requirement `crs` parameter definition . . . 143
A.10.2. Requirement CRS map success . . . 143
A.11. Conformance Class "Orientation" . . . 143
A.11.1. Requirement `orientation` parameter . . . 143
A.11.2. Requirement orientation response headers . . . 143
A.12. Conformance Class "Custom Projection CRS" . . . 144
A.12.1. Requirement `crs-proj-method` parameter . . . 144
A.12.2. Requirement `crs-proj-params` parameter . . . 144
A.12.3. Requirement `crs-proj-center` parameter . . . 144
A.12.4. Requirement `crs-datum` parameter . . . 145
A.12.5. Requirement custom CRS projection response headers . . . 145
A.12.6. Requirement `/projectionsAndDatums` resource . . . 145
A.12.7. Requirement `/projectionsAndDatums` response . . . 146
A.13. Conformance Class "Collection Map" . . . 146
A.13.1. Requirement collection description links . . . 146
A.13.2. Requirement collection description CRS . . . 146
A.13.3. Requirement collection map operation . . . 146
A.14. Conformance Class "Dataset map" . . . 147
A.14.1. Requirement dataset landing page . . . 147
A.14.2. Requirement dataset description extent . . . 147
A.14.3. Requirement dataset description CRS . . . 147
A.14.4. Requirement dataset map operation . . . 148
A.15. Conformance Class "Styled Map" . . . 148
A.15.1. Requirement styled map links . . . 148
A.15.2. Requirement styled map operation . . . 148
A.16. Conformance Class "PNG" . . . 148
A.16.1. Requirement PNG map content . . . 148
A.17. Conformance Class "JPEG" . . . 149
A.17.1. Requirement JPEG map content . . . 149
A.18. Conformance Class "JPEG XL" . . . 149
A.18.1. Requirement JPEG XL map content . . . 149

A.19. Conformance Class "TIFF" . . . 149
A.19.1. Requirement TIFF map content . . . 149
A.20. Conformance Class "SVG" . . . 150
A.20.1. Requirement SVG map content . . . 150
A.21. Conformance Class "HTML" . . . 150
A.21.1. Requirement HTML map content . . . 150
A.22. Conformance Class "API Operations" . . . 150
A.22.1. Requirement API Operations completeness . . . 150
A.22.2. Requirement API Operation identifiers . . . 151
A.23. Conformance Class "CORS" . . . 151
A.23.1. Requirement CORS . . . 151
Annex B: Examples (informative) . . . 152
B.1. Default map . . . 152
B.2. Collection description . . . 153
B.3. Scaling and subsetting the map . . . 155
B.4. Temporal subsetting . . . 161
B.5. Styled maps . . . 162
B.6. Additional dimensions . . . 164
B.7. Coordinate Reference Systems . . . 168
B.8. Calculations to infer appropriate dimensions . . . 170
B.8.1. Plate Carrée (EPSG:4326) Example . . . 170
B.8.2. World Mercator (EPSG:3395) Example . . . 171
B.9. Calculations to infer appropriate bounding boxes . . . 172
B.9.1. Plate Carrée (EPSG:4326) Example . . . 172
B.9.2. World Mercator (EPSG:3395) Example . . . 174
Annex C: Evolution from OGC Web Map Service . . . 176
C.1. From OGC Web Services to OGC Web APIs . . . 176
C.2. Comparison between *WMS* and *OGC API - Maps* . . . 177
C.2.1. Functionality changes from WMS . . . 177
C.2.2. Correspondence between WMS map metadata and OGC API Standards . . . 178
C.2.3. No equivalent to *GetFeatureInfo* as part of the OGC API - Maps - Part 1 . . . 181
Annex D: Revision History . . . 183
Annex E: Bibliography . . . 184

i. Abstract

The *OGC API - Maps - Part 1: Core* Standard defines a Web API for requesting maps over the Web. A *map* is portrayal of geographic information as a digital representation suitable for display on a rendering device (adapted from OGC 06-042/ISO 19128 OpenGIS® Web Map Server (WMS) Implementation Specification [https://portal.opengeospatial.org/files/?artifact_id=14416]). Implementations of the *OGC API - Maps* Standard are designed for a client to easily:

- Request a visual representation of one or more geospatial data layers in different styles;
- Select by area, time and resolution of interest;
- Change parameters such as the background color and coordinate reference systems.

A server that implements *OGC API - Maps* provides information about what maps are offered. *OGC API - Maps* addresses use cases similar to those addressed by the OGC 06-042/ISO 19128 OpenGIS® Web Map Server (WMS) Implementation Specification [https://portal.opengeospatial.org/files/?artifact_id=14416] Standard.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, maps, API, OpenAPI, OGC API

iii. Preface

This document defines the *OGC API - Maps - Part 1: Core* Standard. Suggested additions, changes and comments on this standard are welcome and encouraged. Such suggestions may be submitted as an issue on the *OGC API - Maps* GitHub repository [https://github.com/opengeospatial/ogcapi-maps/issues].

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.

iv. Security considerations

OGC API — Maps — Part 1 only defines HTTP GET operations. As such the security considerations are limited to those applicable to a "read only" service. However, implementations of OGC API - Maps have to process resource paths and query parameters in a way that cannot be used by a client to inject malicious queries that makes available unforeseen data or even force the server to perform unwanted or dangerous actions. The following paragraphs enumerate some security considerations.

Due to the flexibility in generating maps, implementations of the *OGC API - Maps* that are not

optimized can easily encounter requests that take some time to resolve. If several of these requests are processed simultaneously, the server can become slow or unresponsive. Servers should take advantage of space partitioning structures that guarantee a deterministic maximum amount of data to process, such as the 2D Tile Matrix Set data structure (used in Cloud Optimized GeoTIFF, OGC API - Tiles, GeoPackage, etc.), which results in applying cartographic generalization for regions covering a larger area. Servers should also set and apply reasonable limits (e.g., maximum width, etc.) to prevent Denial of Service attacks.

Some OGC API - Maps deployments may assign different roles to different users that may result in accessing different collections or geographical areas that can be represented as maps. The access control can be described in the OpenAPI definition as discussed in OGC API - Common (https://docs.ogc.org/is/19-072/19-072.html#rc_oas30-security). Servers should take care that all resources in all representations and ways they can be requested (e.g., adding query parameters) are managed consistently.

Another security consideration is that maps generated by an Maps API endpoint may be positionally incorrect and potentially guide users to wrong locations. One possible way this situation could happen is the manipulation by a malicious actor of a projection library installed on a server deployment.

Using HTTPS queries is preferred to using HTTP. The obvious reason is the map response returned by the Maps API endpoint should be only accessible to the requesting user. Another reason is related to the public accessibility of the request itself. Request to a Maps API endpoint reveal geographical extents that might be associated with private or sensitive information. For instance, users commonly visit "home" or "work" or "target places" (such as holiday destinations or churches). These requests can be used to predict personal or private activities.

v. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- UAB-CREAF
- Ecere Corporation
- US Army Geospatial Center
- Esri

vi. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Name	**Affiliation**
Joan Masó *(editor)*	UAB-CREAF
Jérôme Jacovella-St-Louis *(editor)*	Ecere
Jeff Harrison	US Army Geospatial Center
Satish Sankaran	Esri

vii. Legacy

The requirements and conformance classes defined in the Maps API Standard are grounded in the work done by many OGC Members in the different versions of the OGC Web Map Service (WMS). In particular, the work of Jean-François (Jeff) de La Beaujardière [https://www.ogc.org/press-release/dr-jeff-de-la-beaujardiere-receives-ogc-lifetime-achievement-award/] was instrumental in the consolidation of WMS. He participated in the original OGC Testbeds that resulted in WMS 1.0, back in April 2000. He became the editor of all subsequent versions, including WMS 1.1, WMS 1.1.1, WMS 1.3, and ISO 19128:2005.

viii. Acknowledgement

Key work for crafting this OGC Standard was undertaken in the Open-Earth-Monitor Cyberinfrastructure (OEMC) project (https://earthmonitor.org/). The OEMC Project received funding from the European Union's Horizon Europe research and innovation program under grant agreement number 101059548. Another project that provided key research is the All Data 4 Green Deal - An Integrated, FAIR Approach for the Common European Data Space (AD4GD) project (https://www.ad4gd.eu/). This project was co-funding by the European Union's Horizon Europe research and innovation program under grant agreement number 101061001 as well as the United Kingdom and Switzerland. Research on applying OGC API Standards for the European digital twin of the ocean is the Integrated DigitaL Framework for Comprehensive Maritime Data and Information Services (ILIAD) project (https://ocean-twin.eu/). This project received funding from the European Union's Horizon Europe research and innovation program under grant agreement number 101061001.

Initial implementations and an Engineering Report constituting the first draft of this standard were also produced thanks to funding from OGC Testbed 15. The editors also thank Natural Resources Canada for progress on this standard achieved as part of GeoConnections 2020-2021 funded project number GNS20-21IFP02 (*Modular OGC API Workflows*).

Chapter 1. Scope

The *OGC API - Maps - Part 1: Core* Standard (hereafter termed the *Maps API*) specifies operations to distribute maps and map tiles in a manner independent of the underlying data store. The Maps API can be described and documented using the OpenAPI [https://www.openapis.org/] specification and specifies resources for discovering and retrieving maps from a Web API.

Specifically, this *OGC API - Maps* Standard supports the following:

- Discovery operations that allow an implementation instance of the Maps API Standard to be interrogated to determine capabilities and to retrieve information about this distribution of maps. This information includes the API definition (if also implementing *OGC API - Common - Part 1: Core* [https://www.opengis.net/doc/is/ogcapi-common-1/1.0]), as well as metadata about the data layers provided and the Coordinate Reference System(s) supported by the Web API implementation instance:
- Retrieval operations that enable client applications to retrieve a map, using a default or pre-defined style, for an arbitrary geospatial resource, a dataset representing the full content available via the Maps API endpoint, or an individual collection of geospatial data representing part of the dataset.
- Parameters for specifying the background and transparency of the map.
- Parameters for specifying the scale of the map.
- A parameter for specifying the pixel size of the device or medium on which the map is intended to be displayed.
- Parameters for retrieving only a subset of the map.
- A parameter for specifying a particular orientation for the map.
- Parameters for specifying a Coordinate Reference System (CRS) for the map by reference or using a projection operation method (as defined by OGC 18-005r4 Abstract Specification Topic 2 Referencing by Coordinates [https://docs.ogc.org/as/18-005r4/18-005r4.html#100]), parameters for that method and a datum.

Chapter 2. Conformance

Conformance with this standard shall be checked using all the relevant tests specified in Annex A (normative) of this document if the respective conformance URLs listed in Table 1 is present in the conformance response. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site.

The standardization targets of all conformance classes are "Web APIs".

The *OGC API - Maps - Part 1: Core* Standard defines twenty-three **(23)** conformance classes, each corresponding to one requirements class.

The core conformance class is the only mandatory class. All other conformance classes are optional and depend on the *core*.

These classes act as building blocks that should be implemented in combination with other more fundamental classes that provide support for documenting and formally describing a Web API (e.g., OpenAPI), listing supported conformance classes and listing available geospatial resources. Possible alternatives for these fundamental classes are *OGC API - Common Part 1: Core*, *OGC API - Features Part 1: Core* or any other non-OGC classes that provide equivalent or alternative functionality.

The *OGC API - Maps - Part 1: Core* Standard is intended to define a minimal set of functions for a useful API for fine-grained access to retrieve maps. Additional classes may be specified in future parts of the *OGC API - Maps* series to add more capabilities or as vendor-specific extensions.

2.1. Requirements classes defining resources

Requirement Class "Core" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core)

The *Maps API* Core specifies requirements that any Web API implementation instance of the Maps API Standard must implement to claim conformance with the *OGC API - Maps - Part 1: Core* Standard. This requirements class defines an operation to retrieve a map but does not define any query parameters to customize the map.

Example request:

```
.../map
```

Requirement Class "Map Tilesets" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/tilesets)

The *map tilesets* requirements class defines a mechanism to list available map tilesets (sets of tiles resulting from tiling data according to a particular tiling scheme, from OGC 17-083r4 Two Dimensional Tile Matrix Set and Tile Set Metadata [https://www.opengis.net/doc/IS/tms/2.0]) supported by the Web API instance and retrieve tiles (see definition in OGC 17-083r4 [https://www.opengis.net/doc/IS/tms/2.0]) of the map representation.

Example requests:

```
.../map/tiles
.../map/tiles/WorldCRS84Quad
.../map/tiles/WorldCRS84Quad/0/0/0
```

NOTE Despite the use of `⋯/map/tiles` path templates in these examples which may become common in implementations of *OGC API - Maps*, these exact paths are only examples and are not required by *OGC API - Tiles* or by this *Maps API* Standard. Other paths are possible provided they are correctly described in the API description of the Web API and the links between resources.

NOTE The "Custom Projection CRS" requirements class is not included in this sub-section because its primary focus is not defining resources, but the Custom Projection requirements class also specifies a /projectionsAndDatums resource listing values available for the parameters it defines.

2.2. Requirements classes defining parameters

These requirements classes of the *Maps API* define parameters that can be used together with any map resource.

NOTE In the effort to make OGC API Standards more consistent, some parts of these requirements classes are expected to be imported by a future part of OGC API - Common or be registered in the building blocks registry. A future revision of OGC API - Maps could reference them and ensure consistency with the relevant common requirements.

Requirement Class "Background" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/background)

The *background* requirements class defines how to request a particular background color (`bgcolor`) and to specify whether areas of the map with no data are `transparent`.

Example requests:

```
.../map?bgcolor=0x001122
.../map?transparent=false
```

Requirement Class "Collection Selection" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/collections-selection)

The *collection selection* requirements class defines how to list specific geospatial data collection resources from which to retrieve maps.

Example request:

```
.../map?collections=buildings,roads,...
```

Requirement Class "Scaling" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/scaling) The *scaling* requirements class defines how to retrieve a resampled map. This is done by specifying a `width` and/or `height` parameter (in pixels), or a `scale-denominator` parameter.

Example requests:

```
.../map?width=1024&height=512
.../map?scale-denominator=20000
```

NOTE	When implementing both this *scaling* requirements class and the *spatial subsetting* requirements class, the `width` and `height` can only be used for specifying the scale when a `bbox` or `subset` is also used for spatial subsetting, as otherwise they take on a subsetting role. When neither `bbox` nor `subset` is used for spatial subsetting, the `scale-denominator` parameter must be used to specify the scale.

Requirement Class "Display Resolution" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/display-resolution) The *display resolution* requirements class defines how to specify the resolution of the display (the size of a pixel in millimeters) using the `mm-per-pixel` parameter at which the map will be visualized so as to properly interpret scale denominators and apply scale-dependent symbology rules.

Example request:

```
.../map?mm-per-pixel=0.14
```

Requirement Class "Spatial subsetting" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/spatial-subsetting) The *spatial subsetting* requirements class defines how to retrieve a spatial subset of a map using either a `bbox` or `subset` parameter.

Example subsetting requests using bounding box:

```
.../map?bbox=120,30,180,90
.../map?bbox-crs=[EPSG:3857]&bbox=500000,430000,1000000,1000000
```

Example subsetting requests using subset:

```
.../map?subset=Lat(30:90),Lon(120:180)
.../map?subset-crs=[EPSG:3857]&subset=X(500000:1000000),Y(430000:1000000)
```

This class also specifies how to retrieve a subset of a map using a `center` parameter, together with `width` and `height` parameters.

NOTE	When implementing both this *spatial subsetting* requirements class and the *scaling* requirements class, the `width` and `height` can only be used for specifying the scale when a `bbox` or `subset` is also used for spatial subsetting, as otherwise they take on a subsetting role. When neither `bbox` nor `subset` is used for spatial subsetting, the `scale-denominator` parameter must be used to specify the scale.

Example subsetting requests using center point and dimensions:

```
.../map?center=-120,60&width=1024&height=512
.../map?center-crs=[EPSG:3857]&center=750000,70000&width=1024&height=512
```

Requirement Class "Date and Time" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/datetime) The *temporal subsetting* requirements class specifies how to request a temporal subset of the data using the `datetime` parameter, or the `subset` parameter for the `time` dimension.

Example requests:

```
.../map?datetime=2018-02-12T23:20:52Z
.../map?subset=time("2018-02-12T23:20:52Z")
```

Requirement Class "General Subsetting" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/general-subsetting) The *general subsetting* requirements class specifies how to request a subset of dimensions of the data besides the spatial and temporal dimensions using the `subset` parameter. This parameter also implies adopting a consistent way to describe all dimensions of the data in the collection's extent description.

Example request:

```
.../map?subset=atm_pressure_hpa(500)
```

Requirement Class "Coordinate Reference System" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/crs) The *CRS by reference* requirements class defines how to specify the output CRS of the map by referencing a URI (or CURIE) for a CRS definition.

Example request:

```
.../map?crs=[EPSG:3031]
```

Requirement Class "Orientation" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/orientation) The *orientation* requirements class defines how to specify an angle (expressed in degrees) for re-orienting how the map is displayed (`orientation`).

Example orientation request:

```
.../map?orientation=40
```

Requirement Class "Custom Projection CRS" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/

req/projection) The *custom projection CRS* requirements class defines how to specify a custom CRS through a projection, including the coordinate operation method (`crs-proj-method`) and associated parameters (`crs-proj-params`), as well as a datum (`crs-datum`). This class also defines a `crs-proj-center` parameter for facilitating the selection of the most likely parameters to center the projection on an area of interest.

Examples of an orthographic projection request:

```
.../map?
   crs-proj-method=[epsg-method:9840]&
   crs-proj-center=Lat(40),Lon(-120)
```

Examples of a Lambert Conic Conformal projection with two standard parallels request:

```
.../map?
   crs-proj-method=[epsg-method:9802]&
   crs-proj-params=[epsg-parameter:8823](40),[epsg-parameter:8824](90)&
   crs-datum=[epsg-datum:6230]
```

NOTE	This "Custom Projection CRS" requirements class also defines a `/projectionsAndDatums` resource listing values available for the parameters it defines.

2.3. Requirements classes defining origins

Requirement Class "Collection Map" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/collection-map)

The *collection map* requirements class specifies how to retrieve maps from a specific geospatial data resource.

Example request:

```
/collections/buildings/map
```

Requirement Class "Dataset Map" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/dataset-map)

The *dataset map* requirements class specifies how to retrieve maps for a whole dataset potentially made up of multiple geospatial data resources. Any Web API implementing this requirements class must support **dataset** maps following this *OGC API - Maps Part 1: Core* Standard. Dataset maps may combine content from multiple geospatial resources, regardless of whether those are available separately (as maps or otherwise).

Example request:

```
/map
```

Requirement Class "Styled Maps" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/styled-map)

The *styled map* requirements class specifies how to retrieve maps for a styled resource.

Example request:

```
.../styles/night/map
```

2.4. Requirements classes defining representations

Requirements classes for encodings

The *Maps API* Standard does not mandate a specific encoding or format for representing maps. Requirements classes are provided for the following common map formats.

PNG (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/png)

Media type: `image/png`

JPEG (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/jpeg)

Media type: `image/jpeg`

JPEG XL (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/jpegxl)

Media type: `image/jxl`

TIFF (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/tiff)

Media type: `image/tiff`

SVG (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/svg)

Media type: `image/svg+xml`

HTML (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/html)

Media type: `text/html`

The Standard remains flexible and extensible for using other formats that users and providers might need through HTTP content negotiation.

That said, this Standard includes recommendations to support, where practical, HTML.

Requirements Class "API Operations" (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/api-operations)

The *API Operations* requirements class specifies requirements to fully describe the *Maps API* operations and use specific operation identifier suffixes when providing an API definition. This requirements class is intended to be used in conjunction with other conformance classes for a

specific API definition language and/or version, such as the OpenAPI 3.0 requirement class defined in *OGC API - Common - Part 1: Core*, or another eventual requirement class for OpenAPI 3.1.

[rc_api_cors] (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/cors)

The *CORS* requirements class specifies a requirement to implement CORS to support JavaScript clients (e.g. Web Browser applications) from a domain different from the OGC API - Maps endpoint.

All requirements classes and conformance classes described in this Standard are owned by the standard(s) identified.

2.5. Summary of conformance URIs

Table 1. Conformance class URIs

Corresponding requirements class	Conformance class URI
Requirement Class "Core"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core
Requirement Class "Map Tilesets"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/tilesets
Requirement Class "Background"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/background
Requirement Class "Collection Selection"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/collections-selection
Requirement Class "Scaling"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/scaling
Requirement Class "Display Resolution"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/display-resolution
Requirement Class "Spatial subsetting"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/spatial-subsetting
Requirement Class "Date and Time"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/datetime
Requirement Class "General Subsetting"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/general-subsetting
Requirement Class "Coordinate Reference System"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/crs
Requirement Class "Orientation"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/orientation
Requirement Class "Custom Projection CRS"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/projection
Requirement Class "Collection Map"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/collection-map

Corresponding requirements class	Conformance class URI
Requirement Class "Dataset Map"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/dataset-map
Requirement Class "Styled Maps"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/styled-map
PNG	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/png
JPEG	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/jpeg
JPEG XL	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/jpegxl
TIFF	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/tiff
SVG	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/svg
HTML	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/html
Requirements Class "API Operations"	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/api-operations
CORS	https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/cors

Chapter 3. References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- OGC: OGC 19-072, **OGC API - Common - Part 1: Core** https://docs.ogc.org/is/19-072/19-072.html
- OpenAPI Initiative (OAI). **OpenAPI Specification 3.0** [online]. 2020 [viewed 2020-03-16]. The latest patch version at the time of publication of this standard was 3.0.3, available at https://spec.openapis.org/oas/v3.0.3
- ISO and IEC: ISO/IEC 10918-1:1994, **Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines** (1994) https://www.iso.org/standard/18902.html
- ISO and IEC: ISO/IEC 18181-1:2022, **Information technology — JPEG XL image coding system - Part 1: Core coding system** (2022) https://www.iso.org/standard/77977.html
- ISO and IEC: ISO/IEC 18181-2:2021, **Information technology — JPEG XL image coding system - Part 2: File format** (2021) https://www.iso.org/standard/80617.html
- ISO and IEC: ISO/IEC 15948:2004, **Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification** (2004) https://www.iso.org/standard/29581.html
- Adobe development association. TIFF Revision 6.0 Final — June 3, (1992) https://www.itu.int/itudoc/itu-t/com16/tiff-fx/docs/tiff6.pdf
- Scalable Vector Graphics (SVG) v2. W3C Candidate Recommendation - October, (2018) https://www.w3.org/TR/SVG2/
- Internet Engineering Task Force (IETF). RFC 8288: **Web Linking** [online]. Edited by M. Nottingham. 2017 [viewed 2020-03-16]. Available at https://tools.ietf.org/rfc/rfc8288.txt
- WHATWG. **HTML**, Living Standard [online, viewed 2020-03-16]. Available at https://html.spec.whatwg.org/
- OGC: OGC 20-057, OGC API - Tiles - Part 1: Core (2022) Available at https://docs.ogc.org/is/20-057/20-057.html
- OGC: OGC 17-083r4, OGC Two Dimensional Tile Matrix Set and Tile Set Metadata 1.0 (2022) https://docs.ogc.org/is/17-083r4/17-083r4.html
- OGC: OGC: 17-069r4, OGC API - Features - Part 1: Core corrigendum, Available at https://docs.ogc.org/is/17-069r4/17-069r4.html

Due to the delay in releasing *OGC API - Common - Part 2: Geospatial data*, we include *OGC API - Features - Part 1* as a temporary normative reference. When *OGC API - Common - Part 2: Geospatial data* is released, the latter should replace *OGC API - Features* as a normative reference.

Chapter 4. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3 [https://portal.opengeospatial.org/files/?artifact_id=34762]), also known as the 'ModSpec'. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

4.1. coordinate reference system

coordinate system that is related to the real world by a datum (source: ISO 19111)

4.2. coordinate system

set of mathematical rules for specifying how coordinates are to be assigned to points (source: ISO 19111)

4.3. data cube (or datacube)

a multi-dimensional ("n-D") array of values. Sometimes, the term *data cube* is applied in contexts where these arrays are massively larger than the hosting computer’s main memory; examples include multi-terabyte/petabyte data warehouses and time series of image data.

4.4. geographic information

information concerning phenomena implicitly or explicitly associated with a location relative to the Earth (source: ISO 19101)

4.5. height

1. vertical elevation of an object in the real world above a reference point (height above the reference ellipsoid for geographic CRS such as [OGC:CRS84h] and [EPSG:4979]).
2. size in pixels of a viewport along its vertical axis

NOTE	Both the *spatial subsetting* and *scaling* requirements classes make use of the `height` query parameter. In both cases, the parameter refers to the height of the image in pixels, but the effect of specifying a fixed value for the parameter is different in each case (selecting the portion of the map to return in the image vs. resampling the map to the requested image size).

4.6. map

portrayal of geographic information as a digital representation suitable for display on a rendering device (adapted from OGC 06-042)

4.7. map background

areas of the map having no data to represent. They are commonly left transparent or filled with a background color

4.8. portrayal

presentation of information to humans (source: ISO 19117)

4.9. rendering device

medium or apparatus on which the map being retrieved will be displayed, whether temporarily (e.g., computer screen) or permanently (e.g., paper).

4.10. scale

ratio between the size of a map shown in the rendering device (commonly in expressed in mm or inches) and the equivalent geographic bounding box of a map in the reality (commonly in expressed in meters or miles). Commonly, the ratio is expressed as a fraction, where 1 unit in the rendering device is equivalent to `n` units in the reality (in the same units of measure). In this case, `n` is called scale denominator.

4.11. viewport

portion of a rendering device where information is being visualized

NOTE For the purpose of this document, the viewport refers to the portion of the map resource response, such as a digital image, where the actual geospatial content of the map is rendered. It excludes any outside margin portions of the image where layout elements such as titles or a legend may be included, as could be defined by an extension. For a two-dimensional viewport, its dimensions are named horizontal and vertical, corresponding to the width and height of the viewport, respectively.

4.12. width

size of in pixels of a viewport along its horizontal axis

NOTE	Both the *spatial subsetting* and *scaling* requirements classes make use of the `width` query parameter. In both cases, the parameter refers to the width of the viewport in pixels, but the effect of specifying a fixed value for the parameter is different in each case (selecting the portion of the map to return in the image vs. resampling the map to the requested image size).

4.13. Web API

An Application Programming Interface (API) using an architectural style that is founded on the technologies of the Web (source: OGC 17-069r3)

NOTE	See Best Practice 24: Use Web Standards as the foundation of APIs [https://www.w3.org/TR/dwbp/#accessAPIs] (W3C Data on the Web Best Practices) for more detail.

Chapter 5. Conventions

This section provides details of conventions used in this document.

5.1. Identifiers

The normative provisions in this standard are denoted by the URI `https://www.opengis.net/spec/ogcapi-maps-1/1.0`.

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

5.2. Link relations

To express relationships between resources, RFC 8288 (Web Linking) [https://tools.ietf.org/rfc/rfc8288.txt] is used.

The following IANA link relation types [https://www.iana.org/assignments/link-relations/link-relations.xhtml] are used in this document:

- **alternate**: Refers to a substitute for this context.
- **self**: Conveys an identifier for the link's context.
- **item**: The target IRI points to a resource that is a member of the collection represented by the context IRI.
- **preview**: Refers to a resource that provides a preview of the link's context. In the context of this specification, it used to link to a low-resolution preview of the map.
- **service-desc**: Identifies service description for the context that is primarily intended for consumption by machines (Web API definitions are considered service descriptions).
- **service-doc**: Identifies service documentation for the context that is primarily intended for human consumption.
- **stylesheet**: Refers to a stylesheet. In the context of this specification, it refers to a cartographic stylesheet, as per the *OGC Styles & Symbology* conceptual model and as provided by *OGC API - Styles*.

In addition, the following link relation type is used for which no applicable registered link relation type could be identified:

- **https://www.opengis.net/def/rel/ogc/1.0/map**: Refers to a resource that is map representation of another geospatial resource (e.g., a collection).
- **https://www.opengis.net/def/rel/ogc/1.0/legend**: Refers to a legend for the map.

Used in combination with *OGC API - Tiles - Part 1: Core*, other link relation types will be used, including:

- **https://www.opengis.net/def/rel/ogc/1.0/tilesets-map**: The target IRI points to a resource that

describes how to provide tile sets of the context resource in map format.

Used in combination with *OGC API - Styles - Part 1: Core*, other link relation types will be used, including:

- **https://www.opengis.net/def/rel/ogc/1.0/styles**: The target IRI points to a resource that describes how to provide styles of the context resource that internally can contain links to maps.

Used in combination with *OGC API - Features - Part 1: Core* or *OGC API - Common - Part 1: Core*, other link relation types will be used, including:

- **https://www.opengis.net/def/rel/ogc/1.0/conformance**: Refers to a resource that identifies the specifications that the link's context conforms to.

Used in combination with *OGC API - Features Part 1: Core* or *OGC API - Common Part 2: Geospatial data*, other link relation types will be used, including:

- **https://www.opengis.net/def/rel/ogc/1.0/data**: Refers to the list of collections available for a dataset.

Each resource representation includes an array of links. Implementations are free to add additional links for all resources provided by the Web API implementing Maps API requirement(s).

5.3. Use of HTTPS

For simplicity, the Maps API only refers to the HTTP protocol. This is not meant to exclude the use of HTTPS and simply is a shorthand notation for "HTTP or HTTPS." Following the recent push towards enhancing web security, public facing web servers are expected to use HTTPS [https://tools.ietf.org/html/rfc2818] rather than HTTP [https://www.ietf.org/rfc/rfc2616.txt]. In the context of Maps API Standard, use of HTTPS provides a layer of security to prevent leaking of information related to users requesting map data for a particular geographical area, which may be private or sensitive in nature (for example, their own location, or the location of endangered species).

Chapter 6. Overview

6.1. Introduction

The *OGC API - Maps* Standard defines building blocks which can be used in a Web API implementation to retrieve geospatial data as maps that are visual portrayals of the data created by applying a style to the data (see Requirement Class "Core"). Maps can present the whole area of the geospatial data or a subset of the content, such as by filtering the data by extent (see Requirement Class "Spatial subsetting"). Maps can be retrieved by arbitrary extent or as tiles (see Requirement Class "Map Tilesets"). The **"Core"** conformance class is the **only mandatory one**. All other conformance classes are optional and depend on the "Core".

An annex with examples of map requests and responses is included as a way to learn by examples how this standard can be applied. See Examples (informative).

An example OpenAPI definition [https://petstore.swagger.io/?url=https://raw.githubusercontent.com/opengeospatial/ogcapi-maps/master/openapi/ogcapi-maps-1.bundled.json] for this Standard is available. Details on how to adjust and bundle the modular and reusable components [https://schemas.opengis.net/ogcapi/maps/part1/1.0/openapi] used in this example OpenAPI definition can be found in [rc_oas3] (Components are also available here [https://github.com/opengeospatial/ogcapi-maps/tree/master/openapi]).

Services and clients are encouraged to support as many of the Coordinate Reference Systems (CRSs) as possible for all geospatial data resources to maximize interoperability. However, this Standard does not require support for any specific CRS.

The *OGC API - Maps* Standard does not specify any requirement for the type of *geospatial data resource* that could be delivered as maps. If the geospatial data resources can be organized into maps, they can be supported regardless of whether they are feature data, coverages, a resource that does not represent data per se (e.g., an annotation) and so forth.

NOTE	Geospatial data resources (e.g., collections) replace the concept of layer in WMS and WMTS. The main difference is that layers in WMS and WMTS were not defined by other OGC API Standards and did not support other functionalities.

These geospatial data resources can advertise one or more map portrayals in several resources, such as the dataset (see Requirement Class "Dataset Map"), a collection (see Requirement Class "Collection Map"), a dataset with a style, or a collection with a style (see Requirement Class "Styled Maps"). Implementations of other OGC API Standards can provide other origins that may be represented as maps. Accessing the *geospatial data resource* content (other than as maps) or its descriptions is possible but out of the scope of this Standard. If a description of the *geospatial data resource* is specified by another standard, and this description has a mechanism to add links to other resources, this Standard indicates the need to add a link to a map of this resource.

The *OGC API - Maps* Standard does not specify how to get a Web API definition, the conformance class list or the collections lists. However, the Maps API standard assumes that the first two are defined by some OGC API Standard (e.g., *OGC API - Common - Part 1: Core* [https://docs.ogc.org/is/19-072/19-072.html]) and the latter by an OGC API for collections (e.g., *OGC API - Common - Part 2: Geospatial*

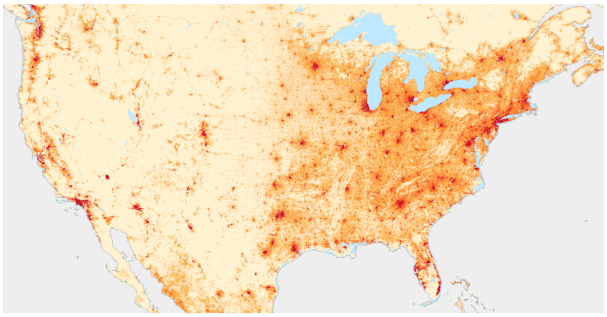
data [https://docs.ogc.org/DRAFTS/20-024.html]). A similar definition is provided directly by *OGC API - Features - Part 1: Core* [https://docs.ogc.org/is/17-069r3/17-069r3.html].

This document is the first part of a series of *OGC API - Maps* "parts" that follows the core and extensions model. Future parts will specify other extensions, such as how to specify cartographic layout elements to be included on the map, how to specify a filter for which elements should be included on the map, and possibly how to query information for a point in a map.

6.2. Map interoperability

One of the most significant use cases for the definition of the *Maps API* is the ability to generate maps on the web from one or more servers that can be combined into a single view. The Maps API Standard provides a clear and common way to request a map image that covers a bounding box of interest with a defined number of pixel rows and columns. When the *Maps API* is implemented by different organizations that offer different collections of information, the same bounding box, in the same CRS, and the same number of rows and columns can be requested from each organization's geospatial repository. The result is a set of images that perfectly overlap showing different variables of the same area depending on the collection of geospatial data requested. For example, a population map can be requested from one organization (e.g., SEDAC Population Density) and weather forecast from another (e.g., NDFD Surface Wind Velocity) in the exact same way allowing to present both together in one overlaid view or side by side.

Table 2. SEDAC GPWv4 Population Density, 2015 and Forecast NDFD, Surface (10m AGL) Wind Velocity (Barb, Knots)



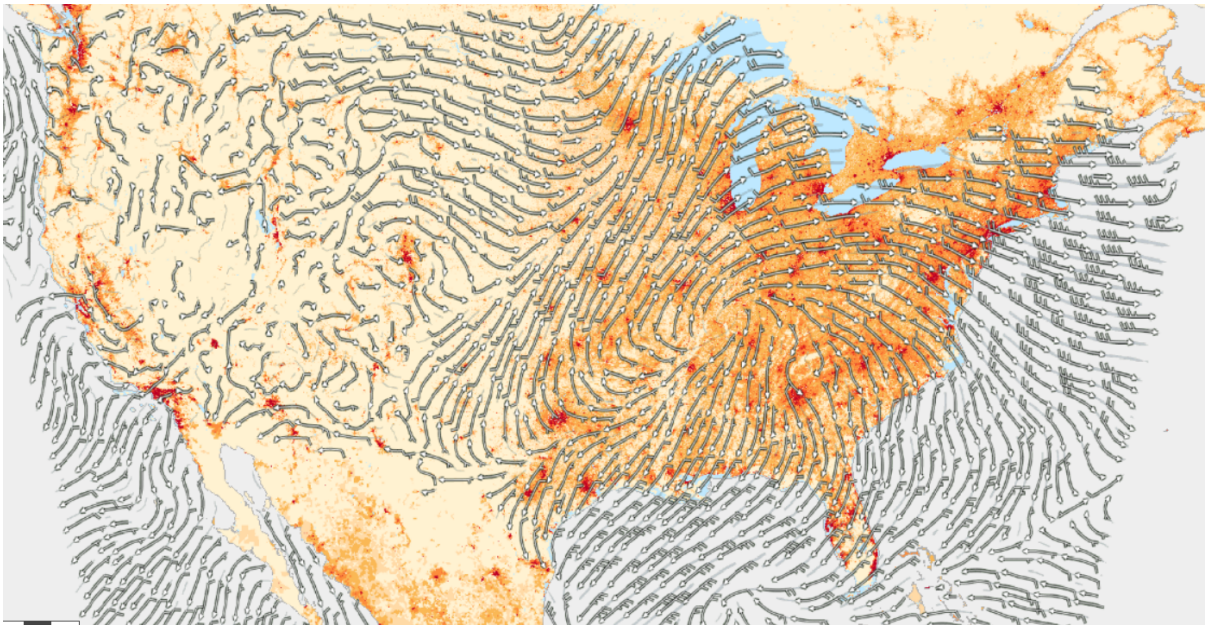


Figure 1. Forecast NDFD wind speed on top of SEDAC Population density

6.3. How to approach an OGC API

There are at least two ways for using an implementation of one or more OGC API Standards.

- Read the landing page, look for links, follow them and discover new links until the desired resource is found
- Read a Web API definition document that specifies a list of paths and path templates to resources.

For the first approach, many resources in a deployed Web API include links with *rel* properties to document the reason and purpose for this relation. The following figure illustrates the resources as ellipses and the links as arrows with the link *rel* as a label.

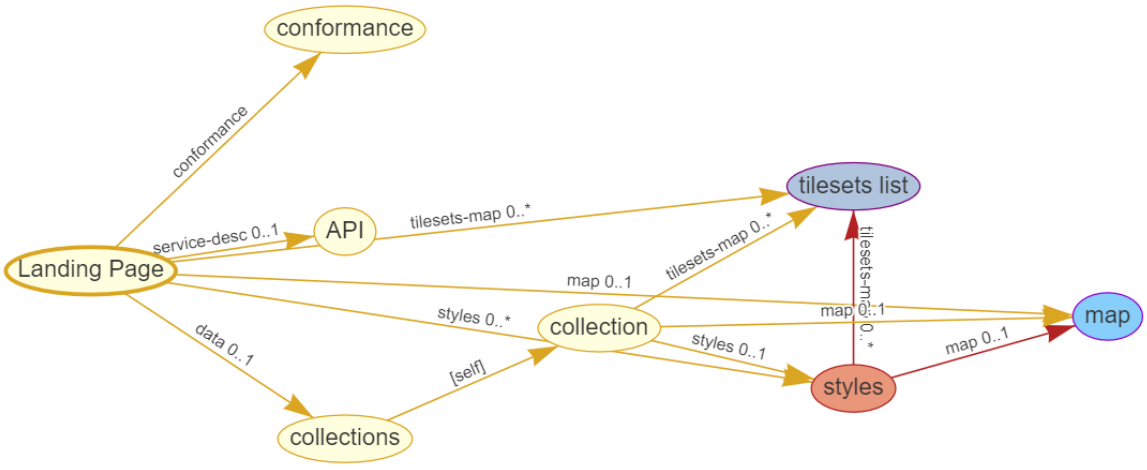


Figure 2. Resources and relations to them via links

For the second approach, implementations should consider the [rc_oas3] which defines the use of *operationID* suffixes, providing a mechanism to associate API paths with the requirements class that they implement.

There is a third way to approach an OGC API implementation instance. This approach relies on assuming a set of predefined paths and path templates. These predefined paths are used in many examples in this document and are presented together in Table 2. It is expected that many implementations of the Maps API Standard will provide a Web API definition document (e.g., OpenAPI) using this set of predefined paths and path templates to get necessary resources directly. All this could mislead the reader into getting the false impression that the predefined paths are enforced. Therefore, building a client that is assuming a predefined set of paths is risky. However, it is expected that many API implementations will follow the predefined set of paths. The clients using this assumption could be successful on many occasions. Again, be aware that these paths are not required by the Maps API Standard.

Table 3. Overview of resources and common direct links that can be used to define an OGC API - Maps implementation

Resource name	**Common path**
Landing page[4]	`{datasetRoot}/`
Conformance declaration[4]	`{datasetRoot}/conformance`
Dataset Maps	
Dataset maps in the default style [1]	`{datasetRoot}/map`
Dataset maps[1,2]	`{datasetRoot}/styles/{styleId}/map`
Dataset map tiles[1,3]	`{datasetRoot}/map/tiles/{tileMatrixSetId}/…`
Geospatial data collections[5]	
Collections[5]	`{datasetRoot}/collections`
Collection[5]	`{datasetRoot}/collections/{collectionId}`
Collection maps in the default style	`{datasetRoot}/collections/{collectionId}/map`
Collection maps[2]	`{datasetRoot}/collections/{collectionId}/styles/{styleId}/map`
Collection map tiles[3]	`{datasetRoot}/collections/{collectionId}/map/tiles/{tileMatrixSetId}/…`

[1] From the whole dataset or one or more geospatial resources or collections

[2] Specified in the *OGC API - Styles* Standard

[3] Specified in the *OGC API - Tiles Part 1: Core* Standard

[4] Specified in the *OGC API - Common Part 1: Core* Standard

[5] Specified in the *OGC API - Common Part 2: Geospatial data* Standard

NOTE	Even though full path and full path templates in the previous table may be used in many implementations of the *OGC API - Maps* Standard, these exact paths are ONLY examples and are NOT required by this Standard. Other paths are possible if correctly described in by the Web API definition document and/or the links between resources.

6.4. *OGC API - Maps* within the OGC API family

6.4.1. What is a map?

A map is a portrayal of data resulting from applying a style, usually in the form of a 2D image format such as PNG or JPEG, or in presentation formats such as SVG. The way the styling rules for a style are applied to the data to create the portrayal is out of scope of this Standard (see *OGC API - Styles* [https://github.com/opengeospatial/ogcapi-styles], as well as specific styles and symbology standards such as OGC Styles & Symbology [https://github.com/opengeospatial/styles-and-symbology], which address this topic).

6.4.2. Implementing *OGC API - Maps* within a Web API

A map can be delivered as a single static resource (only implementing the Maps API "Core" requirement class), or as a dynamic service able to return different maps for arbitrary extents (implementing "Subsetting" requirement class) and/or at arbitrary scales (implementing "Scaling" requirements class). In addition, a map can also be delivered as tiles by combining *OGC API - Maps* with some *OGC API - Tiles* requirements classes. This approach is defined by the "Map Tilesets" requirements class of this Standard, which also correspond to *map tilesets* described in *OGC API - Tiles* [https://docs.ogc.org/is/20-057/20-057.html], with a *map* being a specific type of data resource for which tiles are provided.

The Maps API Standard defines building blocks that can be combined with other APIs generating or providing access to information having a geospatial component, including the other standards in the OGC API family such as *OGC API - Tiles* and *OGC API - Processes*. The Maps API Standard can be referenced by other standards providing resources that can be offered as maps. For example:

- *OGC API - Tiles* [https://docs.ogc.org/is/20-057/20-057.html] specifies the link relation types to access map tilesets from a dataset or collection. *OGC API - Tiles* can also be used to serve the source data (e.g., vector features or coverage data)
- *OGC API - Styles* [https://docs.ogc.org/DRAFTS/20-009.html] defines paths to list available styles from which maps can also be accessed.
- *OGC API - Processes - Part 3: Workflows and Chaining* [https://docs.ogc.org/DRAFTS/21-009.html] provides a mechanism to trigger localized processing workflows as a result of retrieving maps (for a specific area and resolution of interest).

The origin resources to which the map resource can be attached, such as the dataset landing page (defined by *OGC API - Common - Part 1* [https://docs.ogc.org/is/19-072/19-072.html]) and collection (defined by *OGC API - Common - Part 2* [https://docs.ogc.org/DRAFTS/20-024.html]), may also provide access to the data used to generate the maps, alongside the Maps API capability. For example:

- *OGC API - Tiles* [https://docs.ogc.org/is/20-057/20-057.html] also specifies link relation types to access tilesets of vector and coverage data from a dataset or collection.
- *OGC API - Features* [https://docs.ogc.org/is/17-069r3/17-069r3.html] defines an API to access collections of vector features at `/collections/{collectionId}/items` and individual features at `/collections/{collectionId}/items/{itemId}`, including both geometry and properties.
- *OGC API - Coverages* [https://docs.ogc.org/DRAFTS/19-087.html] defines an API to efficiently access information organized as multi-resolution and multi-dimensional datacubes at `/collections/{collectionId}/coverage`. Several common parameters in Coverages API are shared with this Maps API. For some request formulations, it is possible to simply toggle between `/map` and `/coverage` (while keeping the same parameters) to alternate between retrieving the raw data values (a.k.a. a coverage) or a server-side visualization (a.k.a. a map).
- *OGC API - EDR* [https://docs.ogc.org/is/19-086r6/19-086r6.html] defines an API to retrieve spatiotemporal information using multiple query patterns such as cubes, trajectory, and corridors.

The possibilities are endless. For example, a generic open data API giving access to tables, some of them with columns storing latitude and longitude, could be enhanced with OGC API endpoints to provide mapping capabilities.

6.4.3. Dynamic and scalable map viewers

In the OGC, the concept of a map as an image was formulated in 1998 as part of the OGC Web Map Service [https://portal.ogc.org/files/?artifact_id=14416] standards work. At that time, the web was very young. Most HTML pages were static, and JavaScript was a rudimentary programming language capable of controlling user entries in an HTML form and not much more. In that environment, having a service capable of creating a PNG that could be embedded as a HTML page by using an IMG tag provided the first approach to static maps on the web. Replacing the source (SRC) of the IMG tag programmatically with JavaScript, as a reaction of some user actions, provided the first approach to dynamic maps.

The WMS *GetFeatureInfo* request added a limited capability for queryable maps. However, users are now used to moving around the map by frequently doing zoom and pan operations. If the server does not provide a very fast response, the user experience is not smooth and the map display application is perceived as not responsive enough. One possible approach to solve this problem is dividing the viewport into tiles and requesting them separately. Since tiles follow a tile matrix pattern, they can be pre-rendered in the server or cached in the Internet intermediate services. For implementing fast dynamic maps, the *OGC API - Maps* requirement should be combined with *OGC API - Tiles* requirements.

6.4.4. Client-side maps versus server-side maps

The *OGC API - Maps* Standard deals with maps that are generated by the server. The client can present them with no modification. Currently, even the smallest rendering device supports hardware rendering - the transformation from geometries to pixels can be done by the GPU. Transmitting geometries from the server commonly requires less bandwidth than transmitting the rendered map from the server and offers more flexibility on the client-side to personalize the portrayal style. Because of this, it is expected that *OGC API - Maps* use cases will focus more on

static maps, infrequently changing requests for dynamic maps, as well as print cartography, whereas requesting raw data values using *OGC API - Tiles* (e.g., Vector and Coverage Tiles) is better suited for interactive clients presenting dynamic maps.

6.5. Description of the domain

The Maps API Standard defines how to describe the domain of the maps, including spatiotemporal axes as well as additional dimensions.

With the *Collection Map* requirements class, the collection description [https://github.com/opengeospatial/ogcapi-maps/blob/master/openapi/schemas/common-geodata/collectionInfo.yaml] inherited from *OGC API - Common - Part 2* contains an `extent` property that can describe both the spatial and temporal domain of the data. In addition, the *Unified Additional Dimensions* common building block, specified in the *General Subsetting* requirements class and used in the example OpenAPI definition [https://github.com/opengeospatial/ogcapi-maps/blob/master/openapi/schemas/common-geodata/extent-uad.yaml], requires that additional dimensions be described in a similar way to the temporal dimension. This allows providing an overall lower and upper bound (the first `interval` elevement), as well as optional sparse inner intervals where data is found along each dimension (additional `interval` elements). A `grid` property also supports the description of regular and irregular grids. The `resolution` (the distance between any two neighboring cells, an absolute value) and the number of cells (`cellsCount`) can be specified for each regular dimension. A list of `coordinates` where data is found can be specified for irregular dimensions. In addition, the minimum and maximum cell size (`minCellSize` and `maxCellSize`) and equivalent scale denominators (`minScaleDenominator` and `maxScaleDenominator`) can be specified in the collection resource.

The *Dataset Map* requirements class specifies the addition of an `extent` property to the landing page (root resource of the API) of *OGC API - Common - Part 1* based on the same schema as for the collection.

6.6. Subsetting and scaling the map

The Maps API Standard core class provides a way to retrieve the map that is modified by other classes allowing for subsetting the domain, specifying a particular size for the output map image, and changing the default assumption about the physical size of a pixel on the rendering device. The combination of these parameters also define the scale of the map, which affects how scale-dependent symbology rules should be applied. These classes (Scaling, Display resolution and Subsetting) define the following parameters interacting with each other (in a not so trivial manner):

Table 4. Parameters for scaling and subsetting

Parameter	Definition
`width`	Width of the viewport in pixel units
`height`	Height of the viewport in pixel units
`scale-denominator`	Number of units in the physical world that is equivalent to 1 unit on the rendering device
`mm-per-pixel`	Size of one pixel on the rendering device expressed in millimeters. The default value is 0.28 mm

Parameter	Definition
`bbox` (`bbox-crs`) (and the equivalent `subset` and `subset-crs`)	Bounding box of the requested map in CRS coordinates. It defines the geographic size.
`center` (`center-crs`)	Center of the requested map in CRS coordinates. `center` and `bbox` are mutually exclusive.

All these parameters are optional. The server needs to know the geographic extent covered by the map in physical world units, and the size of the map as rendered on the viewport (in both pixel units and physical units). Some combinations completely define both sizes. Some combinations of parameters generate impossible situations and will result in an error. Other combinations require that the server decides a default value for some parameters not provided to be able to resolve the requested sizes. The Maps API Standard only specifies the default value for `mm-per-pixel` leaving to the server freedom to decide about the other parameters. The following tables present an overview of the different combinations possible depending on whether the *Scaling*, *Subsetting* or both *Scaling* and *Subsetting* requirements classes are supported by the implementation, to clarify the relationship between these parameters and provide centralized guidance for implementers.

NOTE The parameter `mm-per-pixel` is not included in these tables but is used for computing one of the `scale-denominator`, dimensions (`width` and `height`), or spatial extent (`bbox`), based on the default or provided values for the others. If not provided in the request, the default is 0.28 mm per pixel.

NOTE Every time that `bbox` appears as a provided parameter in these tables, it represents either `bbox` or the equivalent `subset`.

NOTE Wherever `width` and `height` appear together in these tables, it also represents either of them being specified without the other. Depending on the parameter combination, the server either computes the appropriate value of the omitted dimension so as to reflect the correct scale (when a bounding box is also provided — see relevant requirements and guidance), or uses a default value which is either fixed or tied by a default aspect ratio to the one dimension specified (see recommendation).

Table 5. Always valid requests (no scaling or subsetting parameter)

Parameters provided in the request	Server or resource defaults used	Computed
none	`bbox`, `scale-denominator`, `center`, `width` and `height`	*None*

Table 6. Always invalid parameter combinations

Parameters provided in the request	Explanation
`bbox`, `scale-denominator`, (`width` or `height`)	*Error (conflicts with default or provided* `mm-per-pixel`*)*

Parameters provided in the request	Explanation
`bbox` **and** `center` (with or without additional parameters)	*Error (`bbox` and `center` are mutually exclusive)*

Table 7. Parameter combinations for implementations supporting Subsetting, but not Scaling

Parameters provided in the request	Server or resource defaults used	Computed
`width` and `height`	`scale-denominator` and `center`	`bbox`
`bbox`	`scale-denominator`	`center`, `width` and `height`
`center`	`scale-denominator`, `width` and `height`	`bbox`
`center`, `width` and `height`	`scale-denominator`	`bbox`
`scale-denominator` [1]	`center`	`bbox`, `width` and `height`
`scale-denominator` [1] and `center`	*None*	`bbox`, `width` and `height`
`scale-denominator`, `width` and `height`	*Error (would require rescaling the map)*	
`bbox`, `width` and `height`	*Error (would require rescaling the map)*	
`bbox` and `scale-denominator`	*Error (would require rescaling the map)*	
`scale-denominator`, `center`, `width` and `height`	*Error (would require rescaling the map)*	

[1] The `scale-denominator` parameter is defined in the *Scaling* requirements class. However, an implementation supporting only *Subsetting* may (**but is not required to**) still recognize the `scale-denominator` parameter and compute `width` and `height` dimensions accordingly, along with the corresponding bounding box. In this case, a Subsetting-only implementation may not be applying scale-dependent symbolization rules correctly, since it likely would not render the map anew, but simply cut a piece from a pre-rendered map of a default scale. This is not an issue for maps without any scale-dependent symbolization, such as plain imagery.

Table 8. Parameter combinations for implementations supporting Scaling, but not Subsetting

Parameters provided in the request	Server or resource defaults used	Computed
`width` and `height`	`bbox` and `center`	`scale-denominator`
`scale-denominator`	`bbox` and `center`	`width` and `height`
`scale-denominator`, `width` and `height`	*Error (would require subsetting the map)*	
`bbox` **or** `center` (with or without additional parameters)	*Error (would require subsetting the map)*	

Table 9. Parameter combinations for implementations supporting both Subsetting and Scaling

Parameters provided in the request	Server or resource defaults used	Computed
`width` and `height`	`scale-denominator` and `center`	`bbox`
`bbox`	`width` and `height`	`scale-denominator` and `center`
`center`	`scale-denominator`, `width` and `height`	`bbox`
`center` ,`width` and `height`	`scale-denominator`	`bbox`
`scale-denominator`	`center`, `width` and `height`	`bbox`
`scale-denominator` and `center`	`width` and `height`	`bbox`
`scale-denominator`, `width` and `height`	`center`	`bbox`
`bbox`, `width` and `height`	*None (fully defined combination[1])*	`scale-denominator` and `center`
`bbox` and `scale-denominator`	*None (fully defined combination[2])*	`center`, `width` and `height`
`scale-denominator`, `center`, `width` and `height`	*None (fully defined combination[2])*	`bbox`

[1] This combination corresponds to the WMS parameters and should be used for obtaining identical results from different implementations.

[2] Different implementations may maintain a slightly different relationship between the dimensions (`width` and `height`), the spatial extent (`bbox`) and the `scale-denominator`, based on different considerations for calculating the scales of the map across each dimension. This may result in the `bbox`, `width` or `height` being computed differently between these implementations. Clients should always use `Content-Bbox:` header to properly georeference the output, and not expect unspecified parameters to be computed to a particular value.

NOTE Changing the output CRS using the `crs` parameter will of course also have an impact on the mapping between pixels on the map and units in the real world, and on the calculated bounding box (in output CRS units).

See examples in an annex for computations infering dimensions and infering bounding boxes based on specified parameters.

6.7. Available formats and map response expectations

The Maps API Standard defines six requirements classes for specific encodings to encode map data. Additional encodings can be supported using HTTP content negotiation, following conventions specific to those encodings.

Chapter 7. Requirement Class "Core"

7.1. Overview

Requirements Class Core	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core	
Target type	Web API
Dependency	

The core requirements class specifies a map resource. This is a resource that represents the geospatial data as a rendered map. Requesting a map resource, as defined in the core, results in retrieving a generic map as a graphical representation of the data in the repository being accessed via the API endpoint. When using an implementation of the core requirements class, the representation is not constrained in any way by the client. Therefore, the server is free to return a total or a partial representation of the resource at an arbitrary size. Additional Maps API requirement classes add query parameters to personalize the server behavior to the client needs. These requirement classes define specific parameters that allow the client to control the map size and resolution of the map (width, height, bounding box and CRS) as well as the background of the map. An additional requirement class defines how a map resource can be retrieved as tiles by applying the OGC API - Tiles.

A map resource can be obtained from an arbitrary geospatial resource by adding `/map` to the resource URI. The Maps API Core does not specify to which resources `/map` is applicable. However, a list of common paths to resources exposed by the API, where maps can commonly be obtained is presented in the following table. The Collection Maps], Dataset maps and Styled Maps requirements classes define some of these possibilities.

Table 10. Common map resources in a geospatial API

Resource Path	**Description**
/map	A map representing dataset behind the API in the default style
/styles/{styleId}/map	A map representing dataset behind the API in the `styleId` style
/collections/{collectionId}/map	A map representing `collectionId` in the default style
/collections/{collectionId}/styles/{styleId}/map	A map representing `collectionId` in the `styleId` style

NOTE There is no mandatory dependency of the Maps API core on *OGC API - Common*. This allows various Web APIs to implement the core requirements class without the need to comply to the OGC API - Common structure (landing page, conformance, API description). However, many implementations may specify a dependency on *OGC API - Common* in their conformance page.

7.2. Map resource

A map distribution of a geospatial resource is a pictorial representation of that resource (map). To create a pictorial representation, a style is added to the data in the geospatial resource by a portrayal engine and following portrayal rules. This style can be internal (default style) or can be governed by the client.

This section defines the core resource of the *OGC API - Maps* Standard: A map representation for a geospatial resource. To keep the core of *OGC API - Maps* simple, the core only includes a mechanism to retrieve a map of the size and for the spatiotemporal extent that the server considers optimal.

7.2.1. Operation

Requirement 1	**/req/core/map-op**
A	Every map SHALL be available as a HTTP GET request to a URI that will be composed of three parts. The first part is the URI of a resource that can be represented as a map (with or without a style path parameter). The second part follows the pattern `/map`. The third part providing the retrieval parameters as needed.
B	Only the resources (e.g., a landing page or individual collections) that advertise the pattern `⋯/map⋯` can be retrieved as maps.

7.2.2. Response

Requirement 2	**/req/core/map-response**
A	A successful execution of a map operation SHALL be a response with a HTTP status code 200.
B	The map response SHALL be in the storage CRS specified in the collection description, or https://www.opengis.net/def/crs/OGC/1.3/CRS84 if none is specified, unless overridden by a specific query parameter (see Requirement Class "Coordinate Reference System").
C	The headers SHALL include the "Content-Crs" header with the URI or the safe CURIEs of the CRS used to render the map, except if the content is in the https://www.opengis.net/def/crs/OGC/1.3/CRS84 CRS.
D	The headers of the response SHALL include a "Content-Bbox" header with the actual geospatial boundary of the rendered map.

E	The "Content-Bbox" coordinates SHALL be in the response CRS (indicated in the "Content-Crs" or https://www.opengis.net/def/crs/OGC/1.3/CRS84 if it is not present) and SHALL contain four comma separated numbers representing the lower-left and upper right corners of the response honoring the CRS coordinates order.
F	The body of a response of a successful operation SHALL contain a map in the negotiated format.

NOTE A CURIE {authority}[-{objectType}]:{id} would correspond to the following OGC URI: https://www.opengis.net/def/{objectType}/{authority}/0/{id}. If -{objectType} is missing, the default object type is *crs*.

Below is an example of the successful HTTP response headers describing a PNG image with 1500 columns and 616 rows in the datum WGS84 and projection UTM zone 31N centered in the Barcelona area.

```
HTTP/1.1 200 OK
Content-Type: image/png
Last-Modified: Fri, 21 Jul 2023 14:37:46 GMT
Content-Crs: <https://www.opengis.net/def/crs/EPSG/0/32631>
Content-Bbox: 347736.47,4546722.45,497736.47,4608322.45
Content-Length: 1198247
```

Figure 3. Example of a successful HTTP response body containing a PNG image captured by Sentinel 2 image. Date: 2023-01-22

Below is another example of the successful HTTP response headers describing a PNG image in the datum WGS84 and lat/long centered in the Barcelona area.

```
HTTP/1.1 200 OK
Content-Type: image/png
Last-Modified: Thu, 27 Jul 2023 14:37:46 GMT
Content-Bbox: 2.1486,41.3674,2.1886,41.4074
```

7.2.3. Recommendations

Recommendation 1	**/rec/core/map-op**
A	In the absence of subsetting parameters, the server SHOULD return the entire map.
B	In absence of the scaling parameter, a reasonable width and height SHOULD be selected that preserves the aspect ratio.

Recommendation 2	**/rec/core/content-crs**
A	Even if the content is in the https://www.opengis.net/def/crs/OGC/1.3/CRS84 CRS the headers SHOULD still include "Content-Crs" header to state the CRS.

Recommendation 3	**/rec/core/legend-thumbnail-style**
A	If a legend representing the symbols, lines and colors in the style applied to the map is available for a style of the map, or for the default map resource, the server SHOULD advertise the legend in the list of styles (for each style element) or in the originating resource for the map (e.g., the landing page or collection description), by providing an extra link with a relation type `rel=https://www.opengis.net/def/rel/ogc/1.0/legend`.
B	The server SHOULD support `minScaleDenominator` and `maxScaleDenominator` query parameters for the legend resource specifying the range of scales applicable to the style. Without these parameters the legend is applicable to all scales.
C	If a thumbnail is available for a style of the map, the server SHOULD advertise the thumbnail in the list of styles (for each style element) or in the originating resource for the map by providing an extra link with a relation type `rel=preview`.

D	If one or more style resources are available to apply the style to the map, the server SHOULD advertise them in the list of styles (for each style element) or in the originating resource for the map by providing extra links with a relation type `rel=stylesheet`.

Recommendation 4	**/rec/core/multiple-media-types**
A	If "image/png" and "image/jpeg" are supported at an endpoint and both are present in the HTTP "Accept" request header with the same `q` value, and no other supported output format is present with a higher `q` value, the server SHOULD select between PNG and JPEG and return the format considered optimal. Typically, PNG will be returned if there is transparency or for categorical maps where a limited number of colors are used, and JPEG otherwise.

Retrieving a map without subsetting or tiling has limited use. An implementation should therefore consider allowing retrieving only part of a map by supporting the *map tilesets* and/or the *spatial subsetting* requirements class.

NOTE	The desired encoding is selected using HTTP content negotiation. In addition to the parameters specified by the Maps API core, other parameters should be added.
NOTE	HTTP content negotiation replaces the `FORMAT=` parameter in WMS. However, for convenience, some implementations can implement a `f=` parameter for circumstances where content negotiation is not possible or available (e.g., testing the API in a web browser, or using a URL embedded in an HTML IMG SRC tag).

7.3. Declaration of conformance classes

To support "generic" clients that want to access implementations of multiple OGC API Standards and extensions - and not "just" a specific API / server, the deployed API must declare the conformance classes it implements and conforms to.

The conformance page mainly consists of a list of links.

Requirement 3	**/req/core/conformance-success**
A	If the API instance has a mechanism to advertise conformance classes, the list of conformance classes SHALL include the ones defined in this standard and listed in Table 1 that are supported by this API implementation instance.

If the server declares conformity also to *OGC API - Common - Part 1: Core* or to *OGC API - Features - Part 1: Core*, then the OGC API requirements for declaring conformance, such as the use of the

`/conformance` resource, must be considered. In the JSON format, the conformance resource is an array of links following the link schema defined in *OGC API - Common - Part 1: Core* or in *OGC API - Features - Part 1: Core*. Below is an example fragment of a conformance information page of an API implementation conformant to *OGC API - Common* and *OGC API - Maps*.

Example 1. Conformance Information Page fragment

```
{
  "conformsTo": [
    "https://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core",
    "https://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections",
    "https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core"
  ]
}
```

Chapter 8. Requirement Class "Map Tilesets"

8.1. Overview

Requirements Class Map Tilesets	
Target type	Web API
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/tilesets	
Dependency	https://www.opengis.net/spec/ogcapi-tiles-1/1.0/req/tilesets-list https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Map Tilesets requirements class describes how to provide access to a map as tilesets in combination with the *OGC API - Tiles* [https://docs.ogc.org/is/20-057/20-057.html] Standard, which is based on the *OGC 2D Tile Matrix Set and Tileset Metadata* [https://docs.ogc.org/is/17-083r4/17-083r4.html] Standard. Tiles facilitate caching of data for clients and servers by requesting predefined subsets at fixed resolutions according to multi-resolution set of grids called a *TileMatrixSet*.

NOTE WMTS was focused on map tiles (tiles that are rendered using a style). Since *OGC API - Tiles* allows for other types of tiled data (e.g., tiled vector data and tiled coverage data), *map tiles* is used to refer to the WMTS 1.0 usage of tiles.

While supporting map tiles and map tilesets without conforming to this requirements class is possible, implementing this requirements class means that the tiles resources support parameters defined by *OGC API - Maps* in the following requirement classes:

- Background (`bgcolor` and `transparent` parameter),
- Display Resolution (`mm-per-pixel`),
- Spatial subsetting (only if a vertical dimension is available, since the 2D Tile Matrix Sets already handle partitioning the other two dimensions, and only for `subset` and `subset-crs` parameters),
- General subsetting (`subset` for dimensions beyond spatial and temporal),
- Scaling (`scale-denomimator`, `width` and `height` which will override the usual `tileWidth` and `tileHeight` of the requested tile in pixels from the tile matrix, but those values are still used for calculating the geospatial bounding box corresponding to each tile),

NOTE *OGC API - Tiles* defines its own equivalent requirements class for temporal subsetting.

8.2. Link from data resource

A map resource can be accessed as tiles if the API implementation instance exposes a link to map tilesets in the resource that is available as map tiles.

Requirement 4	**/req/tilesets/desc-links**

A	The geospatial data resource (e.g., the collection or landing page description's `links` property) SHALL include a link with the href pointing to a list of tilesets provided of this geospatial data resource using rel: `https://www.opengis.net/def/rel/ogc/1.0/tilesets-map`.

Example 2. Fragment of a collection description document with a links array with one item of the array pointing to a list of map tilesets.

```
{
    "links": [
    ...
    {
      "href": "https://data.example.com/collections/buildings/map/tiles",
      "rel": "https://www.opengis.net/def/rel/ogc/1.0/tilesets-map",
      "type": "application/json"
    }
  ]
}
```

8.3. Map Tileset

A tileset contains the necessary metadata to enable a client application to formulate a tile request from a single geospatial data resource.

8.3.1. Operation

The operations to retrieve a list of tilesets, individual tilesets and individual tiles are described respectively in the Tilesets List, TileSet and Core requirements classes of the *OGC API - Tiles* Standard.

The Map Tileset requirements class specifies that the query parameters of the GET request to individual map tiles support additional parameters as defined in the *background*, *display resolution*, *spatial subsetting* (if a vertical dimension is available), *general subsetting*, and *scaling* requirements classes.

Requirement 5	**/req/tilesets/tiles-parameters**
A	Implementations of the Maps API SHALL support parameters specified in the *background*, *display resolution*, *spatial subsetting* (only for `subset` and `subset-crs` using a vertical dimension, if available), *general subsetting*, and *scaling* requirements classes, for map tiles endpoint if the implementation declares conformance to the associated conformance classes.

8.3.2. Response

Successful GET responses to requests for a list of map tilesets, an individual map tileset, and map tiles are described in the corresponding requirements class of the *OGC API - Tiles* Standard. The response to map tiles requests shall also reflect the parameters as specified in the operation requirement.

Chapter 9. Requirement Class "Background"

9.1. Overview

Requirements Class Background	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/background	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Background requirement class describes how to define the background of the map is represented.

9.2. Map operation

The Maps API core requirements class defines how to get a map. The Map operation requirements class specifies parameters supporting customization of the background of the map.

9.2.1. Parameters `transparent` and `bgcolor`

The `transparent` and `bgcolor` parameters indicate how the absence of data will be represented in the map, what color will show underneath any non-opaque layer, and allow for the map to be overlaid with other maps without completely obscuring the lower layers.

Requirement 6	**/req/background/bgcolor-definition**

A	The map operation SHALL support a parameter `bgcolor` to define a background color with the characteristics defined in the OpenAPI Specification 3.0 fragment: `bgcolor:` `  name: bgcolor` `  in: query` `  description:` ` Hexadecimal red-green-blue color value (0-255) or case-insensitive W3C web color name for the background color (default=0xFFFFFF). For a six digit hexadecimal value, the first and second digits specify the intensity of red. The third and fourth digits specify the intensity of green. The fifth and sixth digits specify the intensity of blue.` ` required: false` ` style: form` ` explode: false` ` schema:` ` type: string` ` default: 0xFFFFFF`
B	The map operation SHALL support a `bgcolor` parameter which can be an hexadecimal red-green-blue color value (from 00 to FF, FF representing 255) or a case-insensitive W3C web color name [https://www.w3.org/wiki/CSS/Properties/color/keywords] for the background color of the map (default=0xFFFFFF). For a six-digit hexadecimal value, the first and second digits specify the intensity of red. The third and fourth digits specify the intensity of green. The fifth and sixth digits specify the intensity of blue.
C	If `bgcolor` is not specified, and either `transparent` is set to `false` or the output format cannot encode transparency, the server SHALL use the background color specified by the requested style.

Recommendation 1	**/rec/background/bgcolor-definition**
A	If `bgcolor` is not specified, and either `transparent` is set to `false` or the output format cannot encode transparency and the style applied to the map does not specify a background color, the server SHOULD use a neutral default color such as white or a light shade of gray.

The list of W3C web color names [https://www.w3.org/wiki/CSS/Properties/color/keywords] which can be specified instead of a hexadecimal value is reproduced in the following table for convenience:

Table 11. Named web color enumeration

Name	Hexadecimal	Value (r, g, b)
black	`0x000000`	0, 0, 0
dimGray	`0x696969`	105, 105, 105
dimGrey	`0x696969`	105, 105, 105
gray	`0x808080`	128, 128, 128
grey	`0x808080`	128, 128, 128
darkGray	`0xa9a9a9`	169, 169, 165
darkGrey	`0xa9a9a9`	169, 169, 165
silver	`0xc0c0c0`	192, 192, 192
lightGray	`0xd3d3d3`	211, 211, 211
lightGrey	`0xd3d3d3`	211, 211, 211
gainsboro	`0xdcdcdc`	220, 220, 220
whiteSmoke	`0xf5f5f5`	245, 245, 245
white	`0xffffff`	255, 255, 255
rosyBrown	`0xbc8f8f`	188, 143, 143
indianRed	`0xcd5c5c`	205, 92, 92
brown	`0xa52a2a`	165, 42, 42
fireBrick	`0xb22222`	178, 34, 34
lightCoral	`0xf08080`	240, 128, 128
maroon	`0x800000`	128, 0, 0
darkRed	`0x8b0000`	139, 0, 0
red	`0xff0000`	255, 0, 0
snow	`0xfffafa`	255, 250, 250
mistyRose	`0xffe4e1`	255, 228, 225
salmon	`0xfa8072`	250, 128, 114
tomato	`0xff6347`	255, 99, 71
darkSalmon	`0xe9967a`	233, 150, 122
coral	`0xff7f50`	255, 127, 80
orangeRed	`0xff4500`	255, 69, 0
lightSalmon	`0xffa07a`	255, 160, 122
sienna	`0xa0522d`	160, 82, 45
seaShell	`0xfff5ee`	255, 245, 238
chocolate	`0xd2691e`	210, 105, 30

Name	Hexadecimal	Value (r, g, b)
saddleBrown	0x8b4513	139, 69, 19
sandyBrown	0xf4a460	244, 164, 96
peachPuff	0xffdab9	255, 218, 185
peru	0xcd853f	205, 133, 63
linen	0xfaf0e6	250, 240, 230
bisque	0xffe4c4	255, 228, 196
darkOrange	0xff8c00	255, 140, 0
burlyWood	0xdeb887	222, 184, 135
tan	0xd2b48c	210, 180, 140
antiqueWhite	0xfaebd7	250, 235, 215
navajoWhite	0xffdead	255, 222, 173
blanchedAlmond	0xffebcd	255, 235, 205
papayaWhip	0xffefd5	255, 239, 213
moccasin	0xffe4b5	255, 228, 181
orange	0xffa500	255, 165, 0
wheat	0xf5deb3	245, 222, 179
oldLace	0xfdf5e6	253, 245, 230
floralWhite	0xfffaf0	255, 250, 240
darkGoldenrod	0xb8860b	184, 134, 11
goldenrod	0xdaa520	218, 165, 32
cornsilk	0xfff8dc	255, 248, 220
gold	0xffd700	255, 215, 0
khaki	0xf0e68c	240, 230, 140
lemonChiffon	0xfffacd	255, 250, 205
paleGoldenrod	0xeee8aa	238, 232, 170
darkKhaki	0xbdb76b	189, 183, 107
beige	0xf5f5dc	245, 245, 220
lightGoldenRodYellow	0xfafad2	250, 250, 210
olive	0x808000	128, 128, 0
yellow	0xffff00	255, 255, 0
lightYellow	0xffffe0	255, 255, 224
ivory	0xfffff0	255, 255, 240
oliveDrab	0x6b8e23	107, 142, 35

Name	Hexadecimal	Value (r, g, b)
yellowGreen	`0x9acd32`	154, 205, 50
darkOliveGreen	`0x556b2f`	85, 107, 47
greenYellow	`0xadff2f`	173, 255, 47
chartreuse	`0x7fff00`	127, 255, 0
lawnGreen	`0x7cfc00`	124, 252, 0
darkSeaGreen	`0x8fbc8f`	143, 188, 139
forestGreen	`0x228b22`	34, 139, 34
limeGreen	`0x32cd32`	50, 205, 50
lightGreen	`0x90ee90`	144, 238, 144
paleGreen	`0x98fb98`	152, 251, 152
darkGreen	`0x006400`	0, 100, 0
green	`0x008000`	0, 128, 0
lime	`0x00ff00`	0, 255, 0
honeyDew	`0xf0fff0`	240, 255, 240
seaGreen	`0x2e8b57`	46, 139, 87
mediumSeaGreen	`0x3cb371`	60, 179, 113
springGreen	`0x00ff7f`	0, 255, 127
mintCream	`0xf5fffa`	245, 255, 250
mediumSpringGreen	`0x00fa9a`	0, 250, 154
mediumAquaMarine	`0x66cdaa`	102, 205, 170
aquamarine	`0x7fffd4`	127, 255, 212
turquoise	`0x40e0d0`	64, 224, 208
lightSeaGreen	`0x20b2aa`	32, 178, 170
mediumTurquoise	`0x48d1cc`	72, 209, 204
darkSlateGray	`0x2f4f4f`	47, 79, 79
darkSlateGrey	`0x2f4f4f`	47, 79, 79
paleTurquoise	`0xafeeee`	175, 238, 238
teal	`0x008080`	0, 128, 128
darkCyan	`0x008b8b`	0, 139, 139
aqua	`0x00ffff`	0, 255, 255
cyan	`0x00ffff`	0, 255, 255
lightCyan	`0xe0ffff`	224, 255, 255
azure	`0xf0ffff`	240, 255, 255

Name	Hexadecimal	Value (r, g, b)
darkTurquoise	0x00ced1	0, 206, 209
cadetBlue	0x5f9ea0	95, 158, 160
powderBlue	0xb0e0e6	176, 224, 230
lightBlue	0xadd8e6	173, 216, 230
deepSkyBlue	0x00bfff	0, 191, 255
skyBlue	0x87ceeb	135, 206, 235
lightSkyBlue	0x87cefa	135, 206, 250
steelBlue	0x4682b4	70, 130, 180
aliceBlue	0xf0f8ff	240, 248, 255
dodgerBlue	0x1e90ff	30, 144, 255
slateGray	0x708090	112, 128, 144
slateGrey	0x708090	112, 128, 144
lightSlateGray	0x778899	119, 136, 153
lightSlateGrey	0x778899	119, 136, 153
lightSteelBlue	0xb0c4de	176, 196, 222
cornflowerBlue	0x6495ed	100, 149, 237
royalBlue	0x4169e1	65, 105, 225
midnightBlue	0x191970	25, 25, 112
lavender	0xe6e6fa	230, 230, 250
navy	0x000080	0, 0, 128
darkBlue	0x00008b	0, 0, 139
mediumBlue	0x0000cd	0, 0, 205
blue	0x0000ff	0, 0, 255
ghostWhite	0xf8f8ff	248, 248, 255
slateBlue	0x6a5acd	106, 90, 205
darkSlateBlue	0x483d8b	72, 61, 139
mediumSlateBlue	0x7b68ee	123, 104, 238
mediumPurple	0x9370db	147, 112, 219
blueViolet	0x8a2be2	138, 43, 226
indigo	0x4b0082	75, 0, 130
darkOrchid	0x9932cc	153, 50, 204
darkViolet	0x9400d3	148, 0, 211
mediumOrchid	0xba55d3	186, 85, 211

Name	Hexadecimal	Value (r, g, b)
thistle	`0xd8bfd8`	216, 191, 216
plum	`0xdda0dd`	221, 160, 221
violet	`0x40e0d0`	238, 130, 238
purple	`0x800080`	128, 0, 128
darkMagenta	`0x8b008b`	139, 0, 139
magenta	`0xff00ff`	255, 0, 255
fuschia	`0xff00ff`	255, 0, 255
orchid	`0xda70d6`	218, 112, 214
mediumVioletRed	`0xc71585`	199, 21, 133
deepPink	`0xff1493`	255, 20, 147
hotPink	`0xff69b4`	255, 155, 180
lavenderBlush	`0xfff0f5`	255, 240, 245
paleVioletRed	`0xdb7093`	219, 112, 147
crimson	`0xdc143c`	220, 20, 60
pink	`0xffc0cb`	255, 192, 203
lightPink	`0xffb6c1`	255, 182, 193

Permission 2	**/per/background/bgcolor-alpha**
A	The implementation MAY support a `bgcolor` parameter with eight hexadecimal digits where the first two represent an alpha value, to be used as the opacity value for no data areas (where "FF" is completely opaque and "00" is fully transparent).

Requirement 7	**/req/background/transparent-definition**

A	The map operation SHALL support an optional parameter `transparent` specifying whether to use or not a transparent background with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	
	``
	  name: transparent
	  in: query
	  description:
	    '(defaults to `true` without a `bgcolor` specified,
	but to `false` when a `bgcolor` is used).'
	  required: false
	  style: form
	  explode: false
	  schema:
	    type: boolean
	    default: true
	``

B	The server SHALL interpret `transparent` as a Boolean indicating whether the background of the map should be transparent.
C	If `transparent` is not specified and a `bgcolor` is not specified, the server SHALL assume a value of `true`.
D	If `transparent` is not specified and a `bgcolor` is specified, the server SHALL assume a value of `false`.
E	If `transparent` is `true` and a `bgcolor` is specified, the server SHALL use 0 for the background's opacity.

NOTE	When not specified, the value of `transparent` defaults to `true` when `bgcolor` is not used, but to `false` when used together with `bgcolor`. For formats with an alpha channel, explicitly setting `transparent=true` together with a `bgcolor` allow for defining the color value to use for the RGB channels where there is no data. Zero (0) will be used for the alpha channel. For formats reserving a color to define transparency (color key), the combination supports selecting a color that does not interfere with the actual values and colors in the map.
NOTE	The `opaque` attribute of layer element in the GetCapabilities response in WMS is deprecated and is not specified by the Maps API.

9.2.2. Response

A successful GET response is described in the core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core)

Requirement 8	/req/background/map-success
A	The color of the map in the areas with no data SHALL be exactly the one specified in the `bgcolor`.
B	In case the output format allows it and in the absence of the `transparent` parameter (or if it is `false`), the opacity (alpha value) of the map in the areas with no data SHALL be exactly 100% if `transparent` is `false` or 0% if `transparent` is `true`.
C	If the renderer supports anti-aliasing, at the edges between data and no-data areas, the opacity SHALL be a value between 0% and 100%.

Chapter 10. Requirement Class "Collection Selection"

10.1. Overview

Requirements Class Collections Selection	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/collections-selection	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

In a deployed Web API incorporating elements of the OGC Maps API Standard that provides access to a complex dataset formed by several geospatial data resources, selecting specific sub-resources of interest when requesting data from this dataset can be useful. The Collection Selection requirements class defines how to include a query parameter when requesting a resource (e.g., dataset tiles) to specify which geospatial data resources (e.g., collections) should be used to generate the response.

This requirements class is particularly useful in conjunction with the Requirement Class "Dataset Map" requirements class. However, sub-components of other resource types could be selected using this requirements class. A requirements class equivalent to this one is included in *OGC API - Tiles*.

10.2. Operation

By default, the geospatial data resources included in the dataset maps responses are unspecified but should represent the dataset as a whole (but not necessarily the entire extent of the dataset content). This class adds a mechanism to select the geospatial data resources to be used to generate the derived resources (e.g., maps or map tiles). In practice this enables the capability to generate resources involving more than one geospatial data sub-resource.

10.2.1. Parameter `collections`

Requirement 9	/req/collections-selection/collections-parameter

A	An operation that acts on a resource consisting of multiple geospatial data sub-resources (e.g., a resource derived from a root dataset) SHALL support an optional parameter `collections` with the following characteristics (shown as OpenAPI Specification 3.0 fragment):
	
	`name: collections`
	`in: query`
	`required: false`
	`style: form`
	`explode: false`
	`schema:`
	`  type: array`
	`  items:`
	`  type: string`

B	The parameter `collections` SHALL be supported by maps originating from resources consisting of multiple geospatial data sub-resources that can be addressed by identifiers (e.g. dataset map at `{datasetAPI}/maps/`).
C	Implementations SHALL support a comma-separated list of either geospatial resource identifiers (e.g., collectionId's) and/or full URLs to geospatial resource identifiers.

When this parameter refers to more than one geospatial data resource, this parameter will use the comma (",") as the separator between the resource identifiers in the list. Additional white space will not be used to delimit list items. If a geospatial data resource identifier includes a space or comma, it shall be escaped using the URL encoding rules (IETF RFC 2396).

Permission 3	**/per/collections-selection/valid-collections**
A	An implementation MAY return an error when the specified list of collections is not supported, for reasons such as an incompatible combination, or an unsupported encoding or CRS for some of the selected collections.

10.2.2. Response

A successful GET response is described in the core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core)

Requirement 10	**/req/collections-selection/collections-response**

A	Only the collections of geospatial data enumerated in the values of the `collections` parameter SHALL be used to generate the responses for the resource (map) to which they apply.
B	If there is more than one collection name and the style applied does not specify otherwise, the comma-separated collections SHALL be rendered in the result in an order starting with the first (leftmost) collection and ending with the last (rightmost).

NOTE	In maps and map tiles, sub-requirement B will result in the first collection being portrayed at the "bottom" of the display stack with the others rendered on top of the previous ones, one by one (the rightmost collection will become topmost in the portrayal).

10.2.3. Error conditions

If the value of the parameter `collections` contains a resource id of URI that does not exist on the implementation instance of the Maps API or too many collections are requested, the status code of the response is 400.

If the value of the parameter `collections` has a wrong format or combines one of more geospatial data resources that are not compatible (e.g., they do not have a compatible value specified by other parameters in the request), the status code of the response is 400.

10.3. Service Metadata

The OGC API Common Standard describes a mechanism to expose service-metadata for the API. See: https://docs.opengeospatial.org/DRAFTS/19-072.html#service-metadata-examples

Recommendation 5	**/rec/collection-selection/maxcollections**

A	The service metadata may include a maximum number of collections that can be included to create the map. Clients should not request a map with more collections than this limit and if they to, they should expect that the server will not be able to create a map. These characteristics are defined as (shown as OpenAPI Specification 3.0 fragment).
	
	`x-OGC-limits:`
	`  type: object`
	`  properties:`
	`  maps:`
	`  type: object`
	`  properties:`
	`  maxCollections:`
	`  type: integer`
	`  description:`
	`  Maximum number of collections in the collections parameter. If absent the server imposes no limit.`
	`  example: 5`

B	In the absence of maxCollections the client will assume that there is no limit in the number of collections.

NOTE This mimics the `LayerLimit` element in the WMS 1.3 *GetCapabilities* document.

Chapter 11. Requirement Class "Scaling"

11.1. Overview

Requirements Class Scaling	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/scaling	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Scaling requirement class describes how to scale a map for display by specifying a set of parameters that will define its size (width and height) in pixels of a display device that indirectly defines the scale of the map.

11.2. Map Operation

The Maps API *core* requirements class defines how to retrieve a map. The Scaling requirements class specifies two alternative mechanisms for specifying the scale of the map, either implicitly by providing output dimensions using `width` and `height` parameters, or explicitly by providing a `scale-denominator` parameter.

11.2.1. Parameters `width` and `height`

The width and height parameters indicate the size of the viewport in rows and columns where the map is to be displayed. If the format of the response is a raster image, the number of columns and rows of the responded image will commonly match the width and height of the viewport.

Requirement 11	**/req/scaling/width-definition**

A	The map operation SHALL support a parameter `width` with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	`width:`
	`  name: width`
	`  in: query`
	`  description:`
	`  Width of the viewport in pixel units to present the response (the map subset).`
	`  required: false`
	`  style: form`
	`  schema:`
	`  type: number`

B	The `width` value SHALL be interpreted as the horizontal size (columns) of the viewport where the response will be presented in pixel units (number of pixels).
C	An error SHALL be returned if the `width` number is not a positive integer number.
D	An error SHALL be returned if the value of the `width` exceeds the `maxWidth` property specified in the `x-OGC-limits.maps` object included in the service metadata[1].
E	An error SHALL be returned if the value of the `width` (specified or calculated) times `height` (specified or calculated) exceeds a `maxPixels` property from a `x-OGC-limits.maps` object included in the service metadata[1].
F	An error SHALL be returned if the `width` parameter is used together with the `bbox` (or `subset` for spatial dimensions) as well as the `scale-denominator` parameter.
G	An error SHALL be returned if the `width` parameter is used together with the `scale-denominator` parameter and the implementation does not also support the "Subsetting" requirement class.
H	When the `width` parameter is omitted, the implementation SHALL use an appropriate width which accurately reflects the default or requested scale established as the ratio between the horizontal dimension of the viewport and the corresponding size of the physical world, specifically for the local subset (bounding box) of the map being returned, and taking into consideration the default (0.28 mm/pixel) or specified display resolution (`mm-per-pixel`).

[1] Service metadata may be provided as an extension of the `info` section of the Open API document as indicated in OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html], Annex B.4 and discussed in the last subsection of the Maps API scaling requirements class section.

Requirement 12	**/req/scaling/height-definition**

A	The map operation SHALL support a parameter `height` with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	
	`height:`
	`  name: height`
	`  in: query`
	`  description:`
	`  Height of the viewport in pixel units to present the response (the map subset).`
	`  required: false`
	`  style: form`
	`  schema:`
	`  type: number`

B	The `height` value SHALL be interpreted as the vertical size (rows) of the viewport where the response will be presented in pixel units (number of pixels).
C	An error SHALL be returned if the `height` value is not a positive integer number.
D	An error SHALL be returned if the value of the `height` exceeds the `maxHeight` property specified in the `x-OGC-limits.maps` object included in the service metadata[1].
E	An error SHALL be returned if the value of the `width` (specified or calculated) times `height` (specified or calculated) exceeds a `maxPixels` property from a `x-OGC-limits.maps` object included in the service metadata[1].
F	An error SHALL be returned if the `height` parameter is used together with the `bbox` (or `subset` for spatial dimensions) as well as the `scale-denominator` parameter.
G	An error SHALL be returned if the `height` parameter is used together with the `scale-denominator` parameter and the implementation does not also support the "Subsetting" requirement class.
H	When the `height` parameter is omitted, the implementation SHALL use an appropriate height which accurately reflects the default or requested scale established as the ratio between the vertical dimension of the viewport and the corresponding size of the physical world, specifically for the local subset (bounding box) of the map being returned.

[1] Service metadata may be provided as an extension of the `info` section of the Open API document as indicated in OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html], Annex B.4 and discussed in the last subsection of the Maps API scaling requirements class section

NOTE The `width` and `height` parameters are also defined by the Maps API *spatial subsetting* requirements class. The parameter takes this resampling (scaling) role in requests not using either of the `scale-denominator` parameter (defined in this requirements class), or the `center` parameter (defined in the *spatial subsetting* requirements class), or when the *spatial subsetting* requirements class is not supported. In combination with a bounding box and display resolution (either the default of 0.28 mm/pixel or one specified by `mm-per-pixel` parameter from the *display resolution* conformance class), the `width` and `height` parameters define the scale of the map.

Recommendation 6	**/rec/scaling/dimensions**
A	If the `width` or `height` parameter is not specified for a request, and no specific bounding box is requested (using the `subset` or `bbox` parameters defined in the Subsetting requirement class), the implementation SHOULD use the same value for both dimensions to return a map with a square viewport or preserve a particular width to height ratio.
B	If neither `width` nor `height` parameter is specified for a request, and no specific bounding box is requested (using the `subset` or `bbox` parameters defined in the Subsetting requirement class), the implementation SHOULD select reasonable dimensions that will produce a map that is neither too small, nor require too much bandwidth to transfer or process (for example, around 1000 x 1000 pixels for the default 0.28 mm/pixel resolution).

11.2.2. Parameter `scale-denominator`

A client can specify the desired scale of the map as a scale denominator value using a `scale-denominator` parameter. Given a selected unit of measure, the scale denominator specifies how many of these units in the real world correspond to a distance of one unit on the map as printed or displayed.

NOTE The correspondence between real world units and display units depends on the display resolution for which a default 0.28 mm/pixel is assumed, unless a client specifies otherwise through the display resolution requirements class.

NOTE The `scale-denominator` is the only way to specify scaling when requesting a spatial subset using a `center` point.

Requirement 13	**/req/scaling/scale-denominator-definition**

A	The map operation SHALL support a parameter `scale-denominator` with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	
	``
	  scale-denominator:
	    name: scale-denominator
	    in: query
	    description:
	      Number of units in the real-world corresponding to
	one such unit on the display.
	    required: false
	    style: form
	    schema:
	      type: number
	``

B	The `scale-denominator` value SHALL be interpreted as the number of real-world units corresponding to one of the same unit on the map (as printed or displayed), considering only the local subset of the map being returned, based on the selected (e.g., from display resolution requirements class) or default (0.28 mm/pixel) display resolution.
C	The implementation SHALL establish the correspondence between real-world units and pixel units based on the equation: *physicalMetersPerPixel*= (`mm-per-pixel` / 1000 mm/m) * `scale-denominator`, where the *physicalMetersPerPixel* are not necessarily the same as the CRS units (even if those units are expressed in meters) for the region of that CRS consisting of the map subset being returned.
D	An HTTP 400 error SHALL be returned if the `scale-denominator` parameter is used together with `width` and/or `height` and the implementation does not declare conformance to the *spatial subsetting* requirements class (which specifies that the `width` and `height` parameters can also take on a subsetting role).
E	An HTTP 400 error SHALL be returned if the `scale-denominator` parameter is used together with `width` and/or `height` as well as a `bbox` (or equivalent `subset` parameter for a spatial dimension).

F	If the `scale-denominator` parameter is omitted, the implementation SHALL compute it as needed (for purposes such as applying scale-dependent symbology rules) based on the default or selected dimensions, display resolution, and the spatial subset of the map to return.
G	For implementations also supporting "Subsetting", when the spatial subset of the map is not specified in the request, the `scale-denominator` value (default or specified) SHALL be used to compute this bounding box, taking into consideration the display resolution as well as the default or specified dimensions.

Recommendation 7	**/rec/scaling/symbology**
A	An implementation rendering dynamical maps with styling capabilities SHOULD consider the requested or computed scale in applying scale-dependent symbology rules.

11.2.3. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core)

Recommendation 8	**/rec/scaling/map-success-scale**
A	The resolution of the content represented in the map SHOULD be consistent with the `width` and `height` of the viewport indicated and should consider the display resolution (0.28 mm/pixel, unless the `mm-per-pixel` parameter is used) when applying symbology rules. These considerations should also be applied when simplifying the geometries for a vector format, such as KML or SVG.

11.2.4. Error conditions

A general summary of the HTTP status codes can be found in OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in the OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html] Standard.

If the parameters values for `width` or `height` are out-of-range, the status code of the response will be `413` (Payload too big). If a map is not provided due to lack of data in the area, the status code of the response will be `204` or `404`.

11.3. Scale and aspect ratio considerations (informative)

Since all parameters for map requests are optional, implementations must fill in missing parameters by using default values or computing appropriate values based on the parameters provided by the client and the specifics of the resource for which a map is being retrieved. Implementations supporting the "Scaling" requirement class should compute these values in a way that respects the scale of the map in all dimensions, and thus the aspect ratio of physical features being represented.

For a non-global map, and for implementations supporting both "Scaling" and "Subsetting" where clients can request a subset of the map for a small local region, implementations should consider the **local** scale of the map. For example, the applicable scale at the center, or most equatorial latitude of the map. This is as opposed to the global scale (e.g., at the equator). In addition, implementations should aim for the map scale to apply evenly to all dimensions. Consider a CRS such as Plate Carrée projection (CRS84 or EPSG:4326). When filling in an unspecified `bbox` and/or `width` and `height` parameters, this is possible by applying a different linear scaling for the horizontal and vertical dimensions between the CRS units and the pixel dimensions, based on the most equatorial latitude included in the subset.

Another example is using a World Mercator projection (EPSG:3395). Although the CRS unit is "meters", they are only true meters at the equator. When computing a bounding box extending away from a `center` and a `scale-denominator`, the ratio between the CRS unit meters and true physical meters should be considered in order to accurately reflect the requested scale.

The ideal aspect ratio to maintain is the `physical_width / physical_height = width / height`, whereas `physical_width` and `physical_height` refer to the length of features in the horizontal and vertical dimensions respectively, and `width` and `height` are the dimensions of the returned map in pixels.

In the case of a Plate Carrée CRS, a simple way to accurately reflect the two-dimensional scale is to use the most equatorial latitude contained in the map response as the standard parallel of the equirectangular projection. This effectively implies scaling the horizontal distances of the Plate Carrée projection (assuming a standard parallel at 0°N) units by the cosine of that most equatorial latitude in order to get a more accurate physical horizontal distance.

In the case of a World Mercator CRS, an implementation could compute the number of CRS units separating one degree of longitude at the center latitude and compare that with the actual number of physical meters, based on the `cos(lat)` scaling mentioned above, and the length of a degree in meters (WGS84 semi-major axis of 6,378,137 m * Pi / 180, or 111,319.4907932735805 m).

For slightly more accurate calculations, the WGS84 ellipsoid, can be taken into account using the following polynomial equations (from Wikipedia [https://en.wikipedia.org/wiki/Geographic_coordinate_system#Length_of_a_degree]):

```
metersPerDegLat = 111132.92 - 559.82 * cos(2*lat) + 1.175 * cos(4*lat) - 0.0023 *
cos(6*lat)
metersPerDegLon = 111412.84 * cos(lat) - 93.5 * cos(3*lat) + 0.118 * cos(5*lat)
```

If maintaining the aspect ratio between physical units proves too difficult, implementations could fall back to maintaining the CRS unit aspect ratio instead. This way they ensure that `(maxx - minx)/(maxy - miny) = width / height`, where `minx, miny` are the lower bounds of the output CRS unit space in the returned map, `maxx, maxy` the upper bounds of that output CRS unit space on the map, and `width` and `height` are the dimensions of the returned map in pixels.

When neither width nor height is provided or implied, a good practical approach is to default to a reasonable size which is neither too low resolution, nor requiring too much computation power from the server.

When only one of the two dimension parameters is provided and no specific bounding box is requested, the value of the other could be used to return a square map by default.

When a bounding box is requested, the other dimension should be computed so as to preserve the aspect ratio.

When both dimensions are provided, but no bounding box, those dimensions should be used together with the default or requested center, and the scale denominator should be used to compute the bounding box.

Refer to the guidance for implementers provided in a section of the overview on the interactions of subsetting and scaling parameters. This includes the specific combinations of parameters that can be provided based on which requirement classes are implemented, and which parameter assume default values or should be computed in each case.

When `bbox`, `width` and `height` are all provided, as in OGC Web Map Service (WMS), the client enforces a specific aspect ratio, and these recommendations are not relevant. However, in general circumstances (e.g., 2D representations on a screen), the client should try to provide parameters that preserve the aspect ratio.

CAUTION

Due to the inherent difficulty in implementing the computations correctly for all CRSs and some degree of variability based on how these calculations are performed, implementations may compute missing parameters for the same request differently. Clients should therefore always consider the `Content-Bbox:` response header and the actual dimensions of the returned map when geo-referencing and displaying images returned by the Maps API. For identical responses to the same request across different implementations, clients should always provide a `bbox` (or its `subset` equivalent), `width` and `height` parameters (as in classic WMS requests). This is rather than the `scale-denominator` and `center` parameters, which are primarily intended for convenience directly accessing a Maps API, such as from a Web browser. For the purpose of overlaying map responses from different implementations, a client which does not itself geo-reference the output based on the `Content-Bbox:` response header should always request maps using the `bbox`, `width` and `height` parameters.

See also detailed step-by-step examples in an annex of computations to infer dimensions and infer bounding boxes based on specified parameters.

11.4. Service Metadata

The OGC API - Common Standard describes a mechanism to expose service-metadata for the API. See: OGC API - Common - Part 1: Core [OGC 19-072r1] Annex B.4 (2021)

Recommendation 9	**/rec/scaling/max-width-height**

A	The service metadata may include a minimum and maximum width and height and product of both indicating what is the maximum values that the server will accept to create the map. Clients should not request a map subset with a width or a height greater than these limits and if they to, they should expect that the server will not be able to create a map. These characteristics are defined as (shown as OpenAPI Specification 3.0 fragment):
	
	`x-OGC-limits:`
	`  type: object`
	`  properties:`
	`  maps:`
	`  type: object`
	`  properties:`
	`  maxWidth:`
	`  type: integer`
	`  description:`
	`  Maximum width parameter value. If absent the server imposes no limit.`
	`  example: 2048`
	`  maxHeight:`
	`  type: integer`
	`  description:`
	`  Maximum height parameter value. If absent the server imposes no limit.`
	`  example: 2048`
	`  maxPixels:`
	`  type: integer`
	`  description:`
	`  Maximum  product of width and height parameter values. If absent the server imposes no limit.`
	`  example: 4194304`

C	A Maps API implementation instance may specify the maximum width and height by adding a `minimum` property to the `width` and `height` parameter definition in the OpenAPI as in this OpenAPI Specification 3.0 fragment
	
	`width:`
	`  name: width`
	`  in: query`
	`  description:`
	`  Width of the viewport to present the response in pixel units (the map subset).`
	`  required: false`
	`  style: form`
	`  explode: false`
	`  schema:`
	`  type: number`
	`  minimun: 2048`
	`height:`
	`  name: height`
	`  in: query`
	`  description:`
	`  Height of the viewport to present the response in pixel units (the map subset).`
	`  required: false`
	`  style: form`
	`  explode: false`
	`  schema:`
	`  type: number`
	`  minimun: 2048`

B	In the absence of the `maxWidth` or `maxPixels`, the client will assume that there is no limit in the parameter `width`. In the absence of the `maxHeight` or `maxPixels`, the client will assume that there is no limit in the parameter `height`.

NOTE This mimics the `maxWidth` and `maxHeight` elements in the WMS 1.3 *GetCapabilities* document.

NOTE Service metadata can be embedded in the info section of the OpenAPI instance or in an independent document linked to the landing page with the rel="service-meta"

NOTE

The attributes `maxWidth`, `maxHeight` and `maxPixels` are intended to limit server workload, by providing limitations in the size of the output. If the `width` and `height` limits are defined in relation to the size of a common device screen, Maps API implementations should consider that new devices are being built with more and more pixels and a reasonable limit from the current year may be too restrictive for devices emerging in the next year.

Example of the OpenAPI service metadata section:

```
info:
  title: My Web API
  version: 1.0.0
  description: This example shows population of an OpenAPI Info element with
identifying
    metadata for both the service and the service provider.
  x-OGC-limits:
    maps:
      maxWidth: 2048
      maxHeight: 2048
      maxPixels: 10000000
```

Chapter 12. Requirement Class "Display Resolution"

12.1. Overview

Requirements Class Display Resolution	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/display-resolution	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Display Resolution requirement class describes how to specify an accurate physical size of a pixel for the target display device or medium with a `mm-per-pixel` parameter. This is instead of a default 0.28 mm/pixel and allows applying symbology rules correctly. The physical pixel size is considered for establishing the relationship between the scale of the map and its output dimensions in pixels.

12.2. Map Operation

The Display Resolution requirements class specifies the mm-per-pixel parameter to accurately interpret scale denominators and symbology rules.

12.2.1. Parameter `mm-per-pixel`

By themselves, the width and height of the image to be presented on the display device are not enough to determine the scale of a map. This is because these values are provided in pixel units. Conversely, the `scale-denominator` does not specify the size in pixels of a map for a given spatial area. The scale, being the ratio between the size of features as represented on the map and the physical size of the same features, also depends on the size of display device pixels in physical units. Knowing the size of these pixels supports accurately establishing the physical size of features as they will be presented on the display device and therefore compute the scale correctly by dividing the actual size of the display device in physical units (e.g., millimeters) by the actual size of the extent of the map (e.g., in meters). When this parameter is not specified, a default 0.28 mm/pixel is assumed (as standardized in WMS).

Requirement 14	**/req/display-resolution/mm-per-pixel-definition**

A	The map operation SHALL support a parameter `mm-per-pixel` with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	
	`mm-per-pixel:`
	`  name: mm-per-pixel`
	`  in: query`
	`  description:`
	`  size (in millimeters) of a pixel of the rendering device that presents the response.`
	`  required: false`
	`  style: form`
	`  schema:`
	`  type: number`

B	`mm-per-pixel` SHALL be a positive number indicating the size (in millimeters) of a rendering device pixel.
C	If the parameter `mm-per-pixel` is not provided, the server SHALL assume that the pixel size is 0.28 millimeters (90.71 pixels per inch).

NOTE The definition of 0.28 as a default value is the same value as used in Web Map Service WMS 1.3.0 (OGC 06-042) and in the Symbology Encoding (SE) Implementation Specification 1.1.0 (OGC 05-077r4) that was later adopted by WMTS 1.0 (OGC 07-057r7) and 2D-TMS OGC (17-083r4). 0.28 mm was the actual pixel size of a common display from 2005. This value is still being used as reference, even if current display devices are built with much smaller pixel sizes.

NOTE Since the 1980s, the Microsoft Windows operating system has set its default standard display pixels per inch (PPI) to 96. This value results in an approximate value of 0.264 mm per pixel. The similarity of this value with the actual 0.28 mm adopted in this Standard can create some confusion.

NOTE Modern display devices (screens) have pixels so small that operating systems allow for defining a presentation scale factor bigger than one (e.g. 150%). In these circumstances, the actual size of the device pixels is not the same as the size used by the operating system.

With this parameter, the server has information on the physical size of the map as presented on a display device or printed and can therefore apply styling and symbology rules accordingly. For example, if the server receives a very small value, the server could decide to represent linear elements wider than for normal values, preventing to create a representation of linear features that is going to result in lines that are too thin to be correctly perceived.

12.2.2. Response

A successful GET response is described in the core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core)

Requirement 15	**/req/display-resolution/map-success**
A	For an implementation supporting the Maps API *scaling* requirements class, the implementation SHALL use the `mm-per-pixel` value instead of the default 0.28 mm/pixel when establishing the relationship between the dimensions of the output image, the scale and the spatial extent of the map.
B	The `mm-per-pixel` value SHALL be used instead of the default 0.28 mm/pixel when establishing scale for the purpose of applying styling and symbology rules to the map. For example, this needs to be considered for scale-dependent rule selectors as well as for graphical units in real world units (e.g., meters) or display units (e.g., millimeters).

12.2.3. Error conditions

A general summary of the HTTP status codes can be found in OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in OGC API - Common - Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html] Standard.

If the parameters values for `mm-per-pixel` are out-of-range, the status code of the response will be `400`.

Chapter 13. Requirement Class "Spatial subsetting"

13.1. Overview

Requirements Class Subsetting	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/spatial-subsetting	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Maps API core requirements class defines the map resource. The Maps API core also defines a minimum set of metadata that enables a client application to formulate a map request. The Subsetting requirements class adds the ability to request an arbitrary 2D or 3D spatial subset of a map using either a bounding box specified using one of two equivalent syntaxes: a `bbox` parameter similar to the one in WMS 1.3, or a `subset` parameter similar to WCS 2.1. An alternative possibility is to specify a center point as well as `width` and/or `height` parameters taking into account the scale and the display resolution.

How the third (vertical) spatial dimension is applied to filter the output is left to the implementation and may depend on the nature of the data being represented and/or the negotiated output format. For example, the server could include features located within a vertical interval as part of a 2D map being rendered. In a second example, the extent of a 3D perspective view or a 3D model output would include only the subset of the vertical dimension specified. In a third example, a vertical profile could be returned for a single point or for a path across the other spatial dimensions. In this last example, the subset mechanism would likely only be used to subset the vertical dimension, while another mechanism would be used to select an arbitrary path in the horizontal dimensions (this requirements class does not specify how to provide a path in a map request).

13.2. CRS for subsetting

In WMS 1.3 the CRS parameters had two purposes: Indicate the CRS of the data rendered in the map and the CRS of the coordinates of the `bbox` parameter. In the Maps API Standard, these two objectives have been separated. This section defines the CRS parameters for subsetting.

In this document the CRS for subsetting can have 3 possible values:

- https://www.opengis.net/def/crs/OGC/1.3/CRS84
- `storageCrs` indicated in the collection description.
- the one indicated in the `crs` parameter (the one picked for the map response). This possibility is described in the "CRS" requirement class.

The CRS advertised in the collection's spatial extent is always in https://www.opengis.net/def/crs/OGC/1.3/CRS84 (for Earth centric data and as required by *OGC API - Features*). At least for *OGC API -*

Maps and *OGC API - Coverages*, a native CRS (also called storage CRS) is defined. The `Content-Crs:` header (see Overview) specifies the CRS used in the response to avoid confusion.

NOTE	There is no obligation to provide the maps in https://www.opengis.net/def/crs/OGC/1.3/CRS84.
NOTE	The collection or dataset description will contain a `storageCrs` property indicating the native CRS. If it is not present, then https://www.opengis.net/def/crs/OGC/1.3/CRS84 is assumed. This is specified in the *Collection Map* and *Dataset Map* requirements classes.
NOTE	Providing the extent of the collection or dataset in its native CRS (*storage CRS*) is also required, as specified in the *Collection Map* and *Dataset Map* requirements classes.
NOTE	A list of supported CRSs will be provided by the collection or dataset, as specified in the *Collection Map* and *Dataset Map* requirements classes. The first CRS should be the same as the native CRS (`storageCrs`).

13.2.1. Parameter `bbox-crs`

Requirement 16	**/req/spatial-subsetting/bbox-crs**

A	The map retrieval operation SHALL support a parameter `bbox-crs` with the characteristics defined in the OpenAPI Specification 3.0 fragment
	
	`bbox-crs:`
	`  name: bbox-crs`
	`  in: query`
	`  description: A URI (or safe CURIE) of the coordinate reference system for the coordinates specified in the 'bbox' parameter. The valid values are [OGC:CRS84], the native (storage) CRS (if different), or the output 'crs' (if specified).`
	`  required: false`
	`  schema:`
	`  type: string`
	`  example: https://www.opengis.net/def/crs/OGC/1.3/CRS84`

B	For Earth centric data, the implementation SHALL support https://www.opengis.net/def/crs/OGC/1.3/CRS84 as a value.

C	If the bbox-crs is not indicated https://www.opengis.net/def/crs/OGC/1.3/CRS84 SHALL be assumed.
D	If the storage (native) CRS is known, the storage CRS as a value SHALL be supported. Other conformance classes may allow additional values (see crs parameter definition).
E	The CRS expressed as URIs or as safe CURIEs SHALL be supported.
F	If the `bbox` parameter is not used, the `bbox-crs` SHALL be ignored.

NOTE A CURIE {authority}[-{objectType}]:{id} would correspond to the following OGC URI: https://www.opengis.net/def/{objectType}/{authority}/0/{id}. If -{objectType} is missing, the default object type is *crs*.

13.2.2. Parameter `subset-crs`

Requirement 17	**/req/spatial-subsetting/subset-crs**

A	The map operation SHALL support a parameter `subset-crs` with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	`crs:`
	`  name: subset-crs`
	`  in: query`
	`  description: A URI (or safe CURIE) of the coordinate reference system for the coordinates specified in the 'subset' parameter. The valid values are [OGC:CRS84], the native (storage) CRS (if different), or the output 'crs' (if specified).`
	`  required: false`
	`  schema:`
	`  type: string`
	`  example: https://www.opengis.net/def/crs/OGC/1.3/CRS84`

B	For Earth centric data, https://www.opengis.net/def/crs/OGC/1.3/CRS84 SHALL be supported as a value .
C	If the `subset-crs` is not indicated https://www.opengis.net/def/crs/OGC/1.3/CRS84 SHALL be assumed.

D	If the storage (native) CRS is known, the storage CRS SHALL be supported as a value. Other requirements classes may allow additional values (see crs parameter definition).
E	CRS expressed as URIs or as safe CURIEs SHALL be supported.
F	If no `subset` parameter referring to an axis of the CRS is used, the `subset-crs` SHALL be ignored.

NOTE A CURIE {authority}[-{objectType}]:{id} would map to the following OGC URI: https://www.opengis.net/def/{objectType}/{authority}/0/{id}. If -{objectType} is missing, the default object type is crs.

13.2.3. Parameter `center-crs`

Requirement 18	**/req/spatial-subsetting/center-crs**

A	The map retrieval operation SHALL support a parameter `center-crs` with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	
	`crs:`
	`  name: center-crs`
	`  in: query`
	`  description: A URI (or safe CURIE) of the coordinate reference system for the coordinates specified in the 'center' parameter. The valid values are [OGC:CRS84], the native (storage) CRS (if different), or the output 'crs' (if specified).`
	`  required: false`
	`  schema:`
	`  type: string`
	`  example: https://www.opengis.net/def/crs/OGC/1.3/CRS84`

B	For Earth centric data, https://www.opengis.net/def/crs/OGC/1.3/CRS84 SHALL be supported as a value.
C	If the `center-crs` is not used, https://www.opengis.net/def/crs/OGC/1.3/CRS84 SHALL be assumed.
D	If the storage (native) CRS is known, the storage CRS SHALL be supported as a value. Other requirements classes may allow additional values (see crs parameter definition).

E	CRS expressed as URIs or as safe CURIEs SHALL be supported.
F	If no `center` parameter is used, the `center-crs` SHALL be ignored.

NOTE A CURIE {authority}[-{objectType}]:{id} would map to the following OGC URI: https://www.opengis.net/def/{objectType}/{authority}/0/{id}. If -{objectType} is missing, the default object type is crs.

NOTE For clarification, the default CRS of the `bbox`, `center` and `subset` spatial subsetting parameters is `https://www.opengis.net/def/crs/OGC/1.3/CRS84` and the default output CRS of the map is its native (storage) CRS.

NOTE The `center-crs`, `subset-crs` and `bbox-crs` are defined identically except they are affecting `center`, `subset` and `bbox` parameters respectively.

13.2.4. CURIE permission

Permission 4	**/per/spatial-subsetting/crs-curie**
A	For the `center-crs`, `bbox-crs` and `subset-crs` query parameters, an unsafe CURIE without square brackets MAY be supported by the implementation.

NOTE This makes the notation compatible with WMS.

13.3. Subsetting coordinates

13.3.1. Parameter `bbox`

The `bbox` parameter defines how to subset a map using a bounding box (in `bbox-crs` coordinates). This parameter selects a spatial region consisting of any pixels of the map whose area is contained or intersects with the specified bounding box. Individual pixels (including the ones in the four corners of the map) occupy an area on the ground (unlike a coverage whose cell values represent an infinitely small point). The bounding box describing the spatial content of the map (e.g., in the collection description) will go around the "outside" of the pixels of the map, rather than through the centers of the map's border pixels. This is also true for the description of a coverage whose cells represent a value for an area, which may be available as both *Maps* as well as through *OGC API - Coverages*, allowing to offer them both from the same OGC API collection. For example, a global dataset will always be described using -180 to 180 longitude bounds.

Requirement 19	**/req/spatial-subsetting/bbox-definition**

A	The map operation SHALL support a parameter `bbox` with the characteristics defined in the OpenAPI Specification 3.0 fragment:
	
	`bbox:`
	`  name: bbox`
	`  in: query`
	`  description:`
	`  Bounding box of the rendered map. The bounding box is provided as four or six coordinates`
	
	`  * 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)`
	
	`  The coordinate reference system and axis order of the values are indicated in the `bbox-crs` parameter or if the parameter is missing in https://www.opengis.net/def/crs/OGC/1.3/CRS84`
	`  required: false`
	`  schema:`
	`  type: array`
	`  oneOf:`
	`  - minItems: 4`
	`  maxItems: 4`
	`  - minItems: 6`
	`  maxItems: 6`
	`  items:`
	`  type: number`
	`  format: double`
	`  style: form`
	`  explode: false`

B	`bbox` SHALL be a comma separated list of four or six floating point numbers. If the bounding box consists of six numbers, the first three numbers are the coordinates of the lower bound corner of a three-dimensional bounding box and the last three are the coordinates of the upper bound corner. The axis order is determined by the `bbox-crs` parameter value or longitude and latitude if the parameter is missing (https://www.opengis.net/def/crs/OGC/1.3/CRS84 axis order for a 2D bounding box, https://www.opengis.net/def/crs/OGC/1.3/CRS84h for a 3D bounding box). For example in https://www.opengis.net/def/crs/OGC/1.3/CRS84 the order is left_long, lower_lat, right_long, upper_lat.

C	If the `bbox` parameter is used together with the `center` and/or with a `subset` parameter including any of the dimensions corresponding to those of the map bounding box, the server SHALL return a 400 client error.

NOTE	The `bbox` uses the comma (",") as the separator between the coordinates. Additional white space will not be used to delimit list items. The comma character should not be escaped (IETF RFC 2396).

Example: `bbox=-180,-90,180,90&bbox-crs=[OGC:CRS84]`

Example of a 3D bounding box: `bbox=-180,-90,10000,180,90,12000&bbox-crs=[OGC:CRS84h]`

13.3.2. Parameter `subset`

The `subset` parameter is a common building block used to select a subset of a geospatial resource shared with other OGC API Standards. This can be specified as an interval between lower and higher bound values or as a single value. An example of an interval is "between 10.0 and 20.0 degrees of latitude" `subset=Lat(10:20)`. An example of single value is "150 meters above mean sea level" `subset=h(150)`. The dimensions that can be used within the value specified for this parameter, in the context of this particular requirement class, are the abbreviations for the spatial axes specified in the CRS definition.

The `subset` parameter is defined as follows:

Requirement 20	**/req/spatial-subsetting/subset-definition**

A	The implementation SHALL support a `subset` parameter conforming to the following Augmented Backus Naur Form (ABNF) fragment:
	
	`SubsetSpec: "subset"=axisName(intervalOrSingle)`
	`axisName: {text}`
	`intervalOrSingle: interval | single`
	`interval: low : high`
	`low: single | *`
	`high: single | *`
	`single: {number} | "{text}"`
	
	`Where:`
	`\" = double quote = ASCII code 0x42,`
	`{number} is an integer or floating-point number, and`
	`{text} is some general ASCII text (such as a time and date notation in ISO 8601).`

B	Axis names `Lat` and `Lon` SHALL be supported for geographic CRS and `x` and `y` for projected CRS, which are to be interpreted as the best matching spatial axis in the CRS definition.
C	If a third spatial dimension is supported (if the resource's spatial extent bounding box is three dimensional), the implementation SHALL also support a `h` dimension SHALL also be supported. This is the elevation above the ellipsoid in EPSG:4979 or CRS84h for geographic CRS and `z` for projected CRS, which are to be interpreted as the vertical axis in the CRS definition.
D	A 400 error status code SHALL be returned if an axis name in the subset parameter value does not correspond to one of the axes of the subsetting CRS [2].
E	If the *interval* values fall entirely outside the range of valid values defined for the identified axis, a 204 or 404 status code SHALL be returned.
F	For a CRS where an axis can wrap around, such as subsetting across the dateline (anti-meridian) in a geographic CRS, a *low* value greater than *high* SHALL be supported to indicate an extent crossing that wrapping point.
G	Coordinates SHALL be interpreted as values for the named axis of the CRS specified in the `subset-crs` parameter value or in https://www.opengis.net/def/crs/OGC/1.3/CRS84 (https://www.opengis.net/def/crs/OGC/1.3/CRS84h for vertical dimension) if the `subset-crs` parameter is missing.
H	If the `subset` parameter including any of the dimensions corresponding to those of the map bounding box is used with a `bbox` and/or `center` parameter, the server SHALL return a 400 client error.
I	Multiple `subset` parameters SHALL be interpreted, as if all dimension subsetting values were provided in a single `subset` parameter (comma separated)[1].
	[1] Example: `subset=Lat(-90:90)&subset=Lon(-180:180)` is equivalent to `subset=Lat(-90:90),Lon(-180:180)` [2] Note that this is only valid of the spatial dimensions. The 'additional' dimensions relay on the names of the extent of the collection.

NOTE	A subset parameter for https://www.opengis.net/def/crs/OGC/1.3/CRS84 will read as subset=Lon(left_lon:right_lon),Lat(lower_lat:upper_lat).
NOTE	When the *interval* values fall partially outside of the range of valid values defined by the CRS for the identified axis, the service is expected to return the non-empty portion of the resource resulting from the subset.
NOTE	For the operation of returning a map, there normally is no value in preserving dimensionality, therefore a *slicing* operation (using the *point* notation) is usually equivalent to a *trimming* operation (using the *interval* notation) when the low and high bounds of an interval are the same. Therefore, use of the point notation is encouraged in these cases.

While the processing of the `subset` parameter is specific to the resource and operation for which it is applied, there is a general set of requirements which all implementations must address.

Requirement 21	**/req/spatial-subsetting/subset-response**
A	Only that part of the resource that falls within the bounds of the subset interval or corresponding to the single point value SHALL be returned.
B	If a lower limit of the subset expression is populated with an asterisk "*" THEN the minimum extent of the resource along that axis SHALL be selected.
C	If an upper limit of the subset expression is populated with an asterisk "*" THEN the maximum extent of the resource along that axis SHALL be selected.

Recommendation 10	**/rec/spatial-subsetting/subset-crs-axis-names**
A	The names of the axis SHOULD be the abbreviated names of the axis in the CRS definition (e.g. the ones defined in the EPSG database).
B	'e' (in lowercase), 'X' (lowercase/uppercase) or 'Easting' (lowercase/uppercase) SHOULD be interpreted as synonymous of 'E'.
C	'n' (in lowercase) or 'Y' (lowercase/uppercase) or 'Northing' (lowercase/uppercase) SHOULD be interpreted as synonymous of 'N'.

D	'Long' (lowercase/uppercase) or 'Longitude' SHOULD be interpreted as synonymous of 'Lon'.
E	'Latitude' SHOULD be interpreted as synonymous of 'Lat'.

13.3.3. Parameter `center`, `width` and `height`

The `center` parameter defines the center point of the map in the `center-crs` coordinates from which the `width` and `height` parameters will define a subset, taking into consideration the scale and display resolution of the map.

Requirement 22	**/req/spatial-subsetting/center-definition**
A	A `center` parameter SHALL be supported to specify the center of the subset of the map to include with coordinates in the CRS specified in the `center-crs` parameter value or in https://www.opengis.net/def/crs/OGC/1.3/CRS84 if the `center-crs` parameter is missing.
B	If the `center` parameter is used together with the `bbox` and/or with a `subset` parameter including any of the dimensions corresponding to those of the map bounding box, the server SHALL return a 400 client error.

NOTE This parameter uses the comma (",") as the separator between the coordinates. Additional white space will not be used to delimit list items. The comma character should not be escaped (IETF RFC 2396).

Requirement 23	**/req/spatial-subsetting/width-height**
A	When the `center` parameter and/or the `scale-denominator` parameter is used, or if the *scaling* conformance class is not supported, a `width` and `height` parameter specifying the subset of the map to return around the specified or default center of the map SHALL be supported.
B	The scale of the map SHALL be considered whether returning the map at a native scale or resampled (e.g., using the *scaling* conformance class `scale-denominator` parameter), as well as the display resolution (either the default 0.28 mm/pixel, or the one specified by the `mm-per-pixel` parameter of the *display resolution* conformance class).

NOTE	Although the `scale-denominator` parameter is defined in the "Scaling" requirement class and not in "Spatial Subsetting", implementations supporting only Spatial Subsetting are allowed (but not required) to recognize the parameter to compute `width` and `height` dimensions accordingly, together with a corresponding bounding box.

13.3.4. Scale and aspect ratio considerations when subsetting

For implementations not implementing "Scaling", the map is assumed to always be returned with the same fixed aspect ratio in both its native CRS unit space and pixel space, and the same CRS unit to pixel unit scale regardless of the subset being requested.

For implementations implementing both "Scaling" and "Spatial Subsetting" requirements classes, the scaling and subsetting parameters interact somewhat intricately. Guidance for implementers is provided in a section of the overview on these interactions, including the specific combinations of parameters that can be provided based on which requirement classes are implemented, and which parameter assume default values or should be computed in each case.

The "Scaling" requirement class also includes specific guidance for scale and aspect ratio considerations.

See also examples in an annex of computations to infer dimensions and infer bounding boxes based on specified parameters.

13.4. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core).

Requirement 24	**/req/spatial-subsetting/map-success**
A	The content of the response SHALL represent elements inside or intersecting with the spatial extent of the geographical area of the map identified with the subsetting coordinates.

Normally, the content partially outside the map bounding box will be clipped at the extent of the bounding box. This can be done efficiently when map subsets are in raster format (e.g., map tiles). However, maps containing features in vector format may not clip features that are partially outside to ensure continuity of features or for performance.

Recommendation 11	**/rec/spatial-subsetting/map-outside-bounds**

A	The server should try to honor the subsetting coordinates requested. For example, if the server has no data in the requested area, a map filled with the background color or transparent values based on the relevant query parameters specified should be returned. For example, if the subsetting coordinates requested are partially outside the server data, the area with no data SHOULD be filled with the background color or transparent values based on the relevant query parameters specified.

Permission 5	**/per/spatial-subsetting/map-outside-bounds**
A	When the requested subset is outside the bounds of the requested CRS, the server MAY limit the Content-Bbox to the valid values in the CRS.

NOTE	As defined in the Maps API core requirements class, the `Content-Bbox:` and the `Content-Crs:` response headers report the actual bounding box of the data returned.

13.5. Error conditions

A general summary of the HTTP status codes can be found in OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in OGC API - Common.

If the CRS in the parameter value `bbox-crs`, `subset-crs` or `center-crs` is not supported by the server for this resource, or the parameter value is out-of-range, the status code of the response will be `400`. If the map is not provided due to lack of data in the area, the status code of the response will be `204` or `404`.

Chapter 14. Requirement Class "Date and Time"

14.1. Overview

Requirements Class Temporal Subsetting	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/temporal-partitioning	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Date and Time Subsetting requirements class defines the way date and time can be used as a parameter to filter the content in the map resource.

14.2. Describing the temporal extent

Depending on the type of resource, the way the temporal extent and the resolution of the datetime values that are available for the client to request are described may be different:

- For Collection maps, the collection description should specify the temporal extent of the resources. Maps can be requested inside this extent. If the extent is specified in a way that instant values are provided (e.g., by listing them or by including a resolution) then it be possible to request maps for these instants.
- For Dataset maps, the landing page should specify the temporal extent of the dataset. Maps can be requested inside this extent.

14.3. `datetime` query parameter request and response

The `datetime` parameter is a common building block that is shared with other OGC API Standards.

In the context of this requirements class, the `datetime` parameter selects a time instant or interval of the resource for which a map is requested.

The `datetime` parameter is defined as follows:

Requirement 25	/req/datetime/datetime-definition

A	The `datetime` parameter SHALL have the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: datetime`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

B	Temporal geometries are either a date-time value or a time interval. The parameter value SHALL conform to the following syntax (using ABNF [https://tools.ietf.org/html/rfc5234]):
	
	`interval-closed = date-time "/" date-time`
	`interval-open-start = [".."] "/" date-time`
	`interval-open-end = date-time "/" [".."]`
	`interval = interval-closed / interval-open-start / interval-open-end`
	`datetime = date-time / interval`

C	The syntax of `date-time` is specified by RFC 3339, 5.6 [https://tools.ietf.org/html/rfc3339#section-5.6].
D	Open ranges in time intervals at the start or end are supported using a double-dot (`..`) or an empty string for the start/end.

While the processing of the `datetime` parameter is specific to the resource and operation for which it is applied, there is a general set of requirements which all implementations must address.

Requirement 26	**/req/datetime/datetime-response**
A	If the `datetime` parameter is provided by the client and supported by the server, then only resources that have a temporal geometry that intersects the temporal information in the `datetime` parameter SHALL be part of the result set. If a resource has multiple temporal properties, it is the decision of the server whether only a single temporal property is used to determine the extent or all relevant temporal properties.
B	The `datetime` parameter SHALL match all resources in the collection that are not associated with a temporal geometry.

"Intersects" means that the time (instant or period) specified in the parameter `datetime` includes a timestamp that is part of the temporal geometry of the resource (again, a time instant or period). For time periods this includes the start and end time.

NOTE	ISO 8601-2 (Date and time — Representations for information interchange — Part 2: Extensions) distinguishes open start/end timestamps (double-dot) and unknown start/end timestamps (empty string). For queries, an unknown start/end has the same effect as an open start/end.

Example 3. A date-time

February 12, 2018, 23:20:52 UTC:

`datetime=2018-02-12T23%3A20%3A52Z`

For resources with a temporal property that is a timestamp (like `lastUpdate`), a date-time value would match all resources where the temporal property is identical.

For resources with a temporal property that is a date or a time interval, a date-time value would match all resources where the timestamp is on that day or within the time interval.

Example 4. Intervals

February 12, 2018, 00:00:00 UTC to March 18, 2018, 12:31:12 UTC:

`datetime=2018-02-12T00%3A00%3A00Z%2F2018-03-18T12%3A31%3A12Z`

February 12, 2018, 00:00:00 UTC or later:

`datetime=2018-02-12T00%3A00%3A00Z%2F..`

March 18, 2018, 12:31:12 UTC or earlier:

`datetime=..%2F2018-03-18T12%3A31%3A12Z`

A template for the definition of the `datetime` parameter is available at datetime.yaml [https://beta.schemas.opengis.net/ogcapi/common/part2/0.1/collections/openapi/parameters/datetime.yaml].

14.4. `subset=time` query parameter request and response

The `subset` parameter is a common building block that is shared with other OGC API Standards. The `subset` parameter can be specified as an interval between lower and higher bound values or as a single value. An example of an interval is "between 1st of January 2021 and last day of May 2021" `subset=time("2021-01-01":"2021-05-31")`. An example of single value is "1st of January 2021 at 8am o'clock" `subset=time("2021-01-01T08:00:00")`. The `time` dimension can be used within the value specified for this parameter and should ideally match with the axis abbreviation specified in the CRS definition.

The `subset` parameter is defined as follows:

Requirement 27	**/req/datetime/subset-definition**

A	The `subset` parameter SHALL have the following characteristics (using an Augmented Backus Naur Form (ABNF) fragment):
	
	`SubsetSpec: "subset"="time"(intervalOrSingle)`
	`intervalOrSingle: interval | single`
	`interval: low : high`
	`low: single | *`
	`high: single | *`
	`single: {number} | "{text}"`
	
	`Where:`
	`\" = double quote = ASCII code 0x42,`
	`{number} is an integer or floating-point number, and`
	`{text} is some general ASCII text (such as a time and date notation in ISO 8601).`

B	The implementation SHALL support an axis name "time".
C	The implementation SHALL return a 400 status code if the axis name is not "time" and is not recognized in the context of another requirements class.
D	If the intervalOrPoint values fall entirely outside the range of valid values defined for the identified axis, a 204 or 404 status code SHALL be returned
E	Coordinates SHALL be interpreted as values for the named axis of the CRS specified in the temporal extent, or Gregorian UTC time (as specified in RFC 3339) if it is not specified in the temporal extent.
F	Multiple `subset` parameters SHALL be interpreted as if all dimension subsetting values were provided in a single `subset` parameter (comma separated)[1].
	[1] Example: `subset=Lat(-90:90)&subset=Lon(-180:180)&subset=time("2018-02-12T23:20:52Z")` is equivalent to `subset=Lat(-90:90),Lon(-180:180),time("2018-02-12T23:20:52Z")`

NOTE	When the intervalOrPoint values fall partially outside of the range of valid values defined by the CRS for the identified axis, the service is expected to return the non-empty portion of the resource resulting from the subset.

NOTE	For the operation of returning a map, there normally is no value in preserving dimensionality, therefore a *slicing* operation (using the *point* notation) is usually equivalent to a *trimming* operation (using the *interval* notation) when the low and high bounds of an interval are the same. Therefore, use of the point notation is encouraged in these cases.

While the processing of the `subset` parameter is specific to the resource and operation for which it is applied, there is a general set of requirements which all implementations must address.

Requirement 28	**/req/datetime/subset-response**
A	Only that part of the resource that falls within the bounds of the subset expression SHALL be returned.
B	If a lower limit of the subset expression is populated with an asterisk "*" THEN the minimum extent of the resource along that axis SHALL be selected.
C	If an upper limit of the subset expression is populated with an asterisk "*" THEN the maximum extent of the resource along that axis SHALL be selected.

Requirement 29	**/req/datetime/axis**
A	To subset a generic time dimension, the server SHALL support "time" as axisname in the subset parameter

Example 5. A date-time (subset)

February 12, 2018, 23:20:52 UTC:

```
subset=time(%222018-02-12T23%3A20%3A52Z%22)
```

For resources with a temporal property that is a timestamp (such as `lastUpdate`), a date-time value would match all resources where the temporal property is identical.

For resources with a temporal property that is a date or a time interval, a date-time value would match all resources where the timestamp is on that day or within the time interval.

Example 6. Intervals (subset)

February 12, 2018, 00:00:00 UTC to March 18, 2018, 12:31:12 UTC:

`subset=time(%222018-02-12T00%3A00%3A00Z%22%3A%222018-03-18T12%3A31%3A12Z%22)`

February 12, 2018, 00:00:00 UTC or later:

`subset=time(%222018-02-12T00%3A00%3A00Z%22%3A*)`

March 18, 2018, 12:31:12 UTC or earlier:

`subset=time(*%3A%222018-03-18T12%3A31%3A12Z%22)`

14.5. Actual date & time response header

Recommendation 12	**/rec/datetime/actual-datetime**

A	The server SHOULD add a HTTP header with `OGCAPI-datetime` as a name and a temporal geometry as a value, to indicate the instant or the temporal interval of the content of the resource. The temporal geometries value shall conform to the following syntax (using ABNF [https://tools.ietf.org/html/rfc5234]):
	`interval = instant "/" instant`
	`datetime = instant / interval`
	The syntax of `instant` is specified by RFC 3339, 5.6 [https://tools.ietf.org/html/rfc3339#section-5.6].

14.6. Closest date & time permission

Permission 6	**/per/datetime/closest**
A	In case the requested map is not available in the exact requested datetime the closest or last previous time for which data is available MAY be returned by the server.

NOTE	An Earth Observation use case where this permission is useful is to allow retrieving a map of the last datetime where imagery is available, considering that a certain geographic area may only be observed at an interval of "every few days" and availability may be irregular and conditioned by clouds.

14.7. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

Requirement 30	/req/datetime/map-success
A	The content of that response SHALL be consistent with the requested datetime.

14.8. Error conditions

A general summary of the HTTP status codes can be found in OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in the OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html] Standard.

If the parameter value for `datetime` is outside the domain of the collections, the status code of the response will be `400`.

If the request returns no data (e.g., in combination with other types of subsetting), the status code of the response will be `204` or `404`.

Chapter 15. Requirement Class "General Subsetting"

15.1. Overview

Requirements Class General Subsetting	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/general-subsetting	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The General Subsetting requirements class defines the way a subset of the map resource in additional dimensions beyond spatial and temporal can be specified using the `subset` parameter. This functionality is defined as common building blocks shared with other OGC API Standards.

15.2. Describing the extent of the additional dimensions

When implementing the General Subsetting requirements class, additional dimensions included in an extent description must use a uniform approach to describe the lower and upper bounds of those dimensions based on the `intervals` property. An example is found in a OGC API collection description response described in OGC API - Common: Part 2 and follows the JSON schema in https://github.com/opengeospatial/ogcapi-maps/blob/master/openapi/schemas/common-geodata/collectionInfo.yaml. This is the same as with the temporal dimension.

Requirement 31	**/req/general-subsetting/uniform-additional-dimensions**
A	Wherever applicable (e.g., the collection or dataset description), the extent of any additional dimension(s) beyond temporal and spatial SHALL be described in the same way as the temporal dimension. For example, using the name of the dimension as a key and an object as value, with that object containing an `interval` property and `crs`, `trs`, or `vrs`, as specified in *extent-uad.yaml* [https://raw.githubusercontent.com/opengeospatial/ogcapi-maps/master/openapi/schemas/common-geodata/extent-uad.yaml].

NOTE The `subset-crs` mentioned in Spatial subsetting can only be used in combination with a spatial `subset`.

15.3. `subset` query parameter

The `subset` parameter can be specified as an interval between lower and higher bound values or as a single value. An example of an interval is "between 500.0 and 1013.0 hPa"

`subset=pressure(500:1013)`. An example of single value is "750 hPa" `subset=pressure(750)`. The dimensions that can be used within the value specified for this parameter are the axis abbreviations specified in the CRS definition. All additional dimensions listed in the extent as valid dimensions to be used with the subset parameter must be supported.

Requirement 32	**/req/general-subsetting/subset-definition**

A	A `subset` parameter with the following characteristics (using an Augmented Backus Naur Form (ABNF) fragment) SHALL be supported:
	
	`SubsetSpec: "subset"=axisName(intervalOrSingle)`
	`axisName: {text}`
	`intervalOrSingle: interval | single`
	`interval: low : high`
	`low: single | *`
	`high: single | *`
	`single: {number} | "{text}"`
	
	`Where:`
	`\" = double quote = ASCII code 0x42,`
	`{number} is an integer or floating-point number, and`
	`{text} is some general ASCII text (such as a time and date notation in ISO 8601).`

B	Any name of the *additional* dimensions in the extent of the collection SHALL be supported as axis name.
C	A 400 error status code SHALL be returned if an axis name does not correspond to the name of one of the *additional* dimensions in the extent of the collection.
D	If the *intervalOrPoint* values fall entirely outside the range of valid values defined for the identified axis, a 204 status code SHALL be returned.
E	For an axis that can wrap around, a *low* value greater than *high* SHALL be supported to indicate an extent crossing that wrapping point.
F	Multiple subset parameters SHALL be interpreted as if all dimension subsetting values were provided in a single `subset` parameter (comma separated)[1].

	[1] Example: `subset=Lat(-90:90)&subset=Lon(-180:180)&subset=atm_pressure_hpa(500)` is equivalent to `subset=Lat(-90:90),Lon(-180:180),atm_pressure_hpa(500)`

NOTE When the intervalOrPoint values fall partially outside of the range of valid values defined by the CRS for the identified axis, the service is expected to return the non-empty portion of the resource resulting from the subset.

NOTE For the operation of returning a map, there normally is no value in preserving dimensionality. Therefore a *slicing* operation (using the *point* notation) is usually equivalent to a *trimming* operation (using the *interval* notation) when the low and high bounds of an interval are the same. Therefore, use of the point notation is encouraged in these cases.

15.4. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

15.5. Error conditions

A general summary of the HTTP status codes can be found in OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in the OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html] Standard.

If the parameter value for the additional dimensions is outside the domain of the collections, the status code of the response will be `400`.

Otherwise, if the request returns no data (e.g., in combination with other types of subsetting), the status code of the response will be `204` or `404`.

Chapter 16. Requirement Class "Coordinate Reference System"

16.1. Overview

Requirements Class Coordinate Reference System	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/crs	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

In WMS 1.3 the CRS parameters had two purposes: To indicate the CRS of the data rendered in the map and the CRS of the coordinates of the BBOX parameter. In the Maps API Standard, these two purposes have been separated. This section describes how to request a map in a specific CRS.

For *OGC API - Maps* and *OGC API - Coverages*, a native CRS (also called a *storage CRS*) is defined. The `Content-Crs:` header (see Overview) specifies the CRS used in the response to avoid confusion.

16.2. Operation

The Maps API core requirements class defines how to retrieve a map. The CRS requirements class specifies parameters needed to retrieve a map in a particular output CRS.

16.2.1. Parameter `crs`

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

Requirement 33	/req/crs/crs-definition
A	The map operation SHALL support a parameter `crs` with the characteristics defined in the OpenAPI Specification 3.0 fragment:

```
  crs:
    name: crs
    in: query
    description: A coordinate reference system of the
map response. A list of all supported CRS values can be
found under the collection metadata.
    required: false
    schema:
      type: string
    example:
https://www.opengis.net/def/crs/OGC/1.3/CRS84
```

B	Any of the CRSs listed in the collection (or collections) description SHALL be supported. If the list of supported CRS is not present, only https://www.opengis.net/def/crs/OGC/1.3/CRS84 SHALL be supported.
C	If the spatial subsetting requirements class is supported, the `bbox-crs` and the `subset-crs` SHALL additionally support value specified in the crs parameter.
D	CRS expressed as URIs or as safe CURIEs SHALL be supported.

NOTE When no crs parameters are specified, please refer to the Maps API core conformance class to know about the default crs.

NOTE A CURIE `{authority}[-{objectType}]:{id}` would map to the following OGC URI: `https://www.opengis.net/def/{objectType}/{authority}/0/{id}`. If `-{objectType}` is missing, the default object type is crs.

See how to determine the native (storage) CRS for the collection and the dataset in Requirement Class "Collection Map" and Requirement Class "Dataset Map" respectively.

NOTE The default CRS of the BBOX is https://www.opengis.net/def/crs/OGC/1.3/CRS84 but the default CRS of the map is the native (storage) CRS

Permission 7	**/per/crs/crs-curie**
A	For the `crs` parameter, unsafe CURIE without square brackets MAY be supported by the implementation.

NOTE This makes the notation compatible with WMS.

16.2.2. Response

Requirement 34	**/req/crs/map-success**
A	The content of that response SHALL be consistent with the requested CRS.

16.2.3. Error conditions

A general summary of the HTTP status codes can be found in OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in the OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html] Standard.

If the parameter value for `crs` is not valid for the collections, the status code of the response will be `400`.

Chapter 17. Requirement Class "Orientation"

17.1. Overview

Requirements Class Orientation	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/orientation	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

This requirements class defines the ability for clients to request a map whose content is rotated by a counterclockwise orientation specified in degrees, resulting in the viewing perspective being rotated by that same orientation in a clockwise direction.

17.2. Operation

17.2.1. Parameter `orientation`

Requirement 35	/req/orientation/orientation
A	If rotation of the map content is desired, an orientation parameter SHALL be supported that specifies the amount by which to rotate a map, expressed as counterclockwise degrees. This results in the viewing perspective being rotated by that same orientation in a clockwise direction.
B	If an `orientation` parameter is not specified, a zero-orientation value SHALL be assumed.
C	The orientation SHALL be applied to the map with the center of the selected spatial subset as the pivot point, or the center of the map if none is specified.
D	If an `orientation` parameter is used together with `subset` or `bbox` spatial subsetting parameter, the counterclockwise orientation SHALL be applied to the four corners of the clipping box associated to that subset, as if the equivalent `center`, `width` and `height` spatial subsetting query parameters were used instead. This avoids leaving empty corners in the final rotated map image.

17.2.2. Response headers

Requirement 36	/req/orientation/response-headers
A	For responses to a map request where the `orientation` query parameter was used, a response header `Content-Orientation: [value in decimal degrees]` corresponding to the orientation of the map SHALL be returned.
B	For responses to map request where the `orientation` query parameter was used, the `Content-Bbox` response header SHALL reflect the bounding box in the map output CRS prior to the orientation being applied.

Example of a request and associated response headers:

```
/map?orientation=40
```

```
Content-Crs: <https://www.opengis.net/def/crs/EPSG/0/3995>
Content-Orientation: 40
Content-Bbox: -10000,-4000,10000,4000
```

17.2.3. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

17.2.4. Error conditions

A general summary of the HTTP status codes can be found in *OGC API - Features - Part 1: Core*, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in the OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html] Standard..

If the parameter value for the orientation is not a valid orientation, the status code of the response will be `400`.

Chapter 18. Requirement Class "Custom Projection CRS"

18.1. Overview

Requirements Class Custom Projection CRS	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/projection	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Custom Projection CRS requirements class defines the ability for clients to specify a custom projection CRS based on parameters defining a projection operation method, values for corresponding parameters, as well as a datum. This class also provides a parameter facilitating the task of selecting a projection center across different projections, for which the relevant latitude and longitude parameters differ. Angles are specified in degrees, whereas identifiers for methods, parameters and datums can be specified as URIs or safe CURIEs. Additionally, this requirements class defines a resource at `/projectionsAndDatums` listing the available operation methods and their associated parameters, as well as the possible datums that can be used as values for these parameters.

18.2. Operation

18.2.1. Parameter `crs-proj-method`

The operation method refers mainly to the equations that relates geographic coordinates to projected coordinates and vice-versa.

Requirement 37	/req/projection/crs-proj-method
A	A `crs-proj-method` parameter supporting selection of a projection operation method SHALL be supported.
B	CURIEs in addition to URI to specify the projection method SHALL be supported.

18.2.2. Parameter `crs-proj-params`

The operation method may have some parameters, such as the standard parallel, to set. This query parameter supports setting those values. In some simple cases this query parameter might not be necessary, and clients can consider using `crs-proj-center` instead.

Requirement 38	/req/projection/crs-proj-params

A	A `crs-proj-params` parameter SHALL be supported that enables selection of one or more value for operation method parameters, with values in between parentheses `( )` following the URI of a parameter, and different parameters separated by value. For example, `crs-proj-params=[epsg-parameter:8823](40),[epsg-parameter:8824](90)`.
B	In addition to URI to specify the projection parameters SHALL be supported.

18.2.3. Parameter `crs-proj-center`

To facilitate the task of centering a custom projection on the area of interest, with parameters varying greatly between projections, the `crs-proj-center` parameter automatically sets the most appropriate parameters for the specified center latitude and longitude.

Requirement 39	**/req/projection/crs-proj-center-definition**
A	A `crs-proj-center` parameter of the form `Lat(centerLat),Lon(centerLon)` SHALL be supported to facilitate the selection of the most relevant projection parameters to center a custom projection.
B	The projection-center `Lat` value SHALL be mapped to the first matching operation method parameter available for the selected operation method of the projection query parameter, in this order: • [epsg-parameter:8832] Latitude of standard parallel (PROJ: *lat_ts*) • [epsg-parameter:8823] Latitude of first standard parallel, if only a single standard parallel can be set — i.e., no [epsg-parameter:8824] parameter is available (PROJ: *lat_ts*) • [epsg-parameter:8801] Latitude of natural origin (PROJ: *lat_0*) • [epsg-parameter:8811] Latitude of projection center (PROJ: *lat_0*)

C	The projection-center `Lon` value SHALL be mapped to the first matching operation method parameter available for the selected operation method of the projection query parameter, in this order: • [epsg-parameter:8802] Longitude of natural origin (PROJ: *lon_0*) • [epsg-parameter:8812] Longitude of projection center (PROJ: *lon_0*) • [epsg-parameter:8833] Longitude of origin (PROJ: *lon_0*)

NOTE For some projections where there is not a clear single center latitude parameter, such as those with multiple standard parallels, clients should use `crs-proj-params` to unambiguously set each parameter instead.

18.2.4. Parameter `crs-datum`

Requirement 40	**/req/projection/crs-datum**
A	A `crs-datum` parameter as a URI allowing to select a datum for the output CRS SHALL be supported.
B	CURIEs in addition to URI to specify the datum parameter SHALL be supported.
C	If a `crs-datum` parameter is not specified, the native (storage) CRS datum SHALL be assumed (the WGS84 ensemble `[epsg-datum:6326]` datum is assumed if the native CRS is not declared).

18.2.5. Response headers

Requirement 41	**/req/projection/response-headers**
A	For responses to map request where the `crs-proj-method` query parameter was used, a response header `Content-Crs-Method: <[URI]>` including the URI of the projection operation method SHALL be returned.
B	A response header `Content-Crs-Method-Params: <URI>=[value]; ⋯` SHALL be returned. This includes the URI of the projection operation parameters and its value for each parameter specified using the `crs-proj-method` or `crs-proj-center` query parameters.

C	For responses to map request where the `crs-datum` query parameter was used, a response header `Content-Crs-Datum: <[URI]>` corresponding to the URI of the projection operation method SHALL be returned.
D	For responses to map requests specifying the `crs-proj-method` query parameter, a `Content-Crs` response header SHALL NOT be included.
E	For responses to map requests specifying the `crs-proj-method` query parameter, the CRS of the `Content-Bbox` response header coordinates SHALL be in the custom CRS defined by this operation method and its parameters.

Example of a request and associated response headers:

```
/map?
   crs-proj-method=[epsg-method:9840]&
   crs-proj-center=Lat(30),Lon(40)&
   crs-datum=[epsg-datum:6326]
```

```
Content-Crs-Method: <https://www.opengis.net/def/method/EPSG/0/9840>
Content-Crs-Method-Params: <https://www.opengis.net/def/parameter/EPSG/0/8801>=30;
<https://www.opengis.net/def/parameter/EPSG/0/8802>=40
Content-Crs-Datum: <https://www.opengis.net/def/datum/EPSG/0/6326>
Content-Bbox: -1000,-800,1000,800
```

18.2.6. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

18.2.7. Error conditions

A general summary of the HTTP status codes can be found in OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as well as in the OGC API – Common – Part 1: Core [https://docs.ogc.org/is/19-072/19-072.html] Standard.

If the parameter value for the parameters is not valid, the status code of the response will be `400`.

18.3. Available projections

18.3.1. Available projections operation

The resource at `/projectionsAndDatums` supports a `GET` operation, with JSON as an available

representation, returning a list of the supported projection operation methods and their associated parameters, as well as a list of available datums, applying to all map retrieval operations of the API.

Requirement 42	**/req/projection/projections-resource**
A	A GET operation at `/projectionsAndDatums` providing at minimum a JSON representation SHALL be supported.

18.3.2. Available projections response

The custom projections resource includes a `methods` property listing the available projections operation methods, corresponding to the valid values for the `crs-proj-method` query parameter, and their associated parameters, corresponding to the valid values for the `crs-proj-params` parameters.

The custom projections resource also includes a `datums` property listing the available datums corresponding to the valid values for the `crs-datum` query parameter.

Requirement 43	**/req/projection/projections-response**
A	The implementation SHALL include in its response for the `/projectionsAndDatums` resource the complete list of custom CRS projection operation methods supported for map retrieval operations.
B	The implementation SHALL include in its response for the `/projectionsAndDatums` resource the complete list of custom CRS datums supported for map retrieval operations.
C	In the JSON representation, the list of supported projection operation methods SHALL be provided as a dictionary (associative array) value for a `methods` property associating operation method objects (including optional `title` and `description` properties) to the corresponding identifiers to be used as values for the `crs-proj-method` query parameter. These operation method identifiers SHALL be safe CURIEs when a registered URI exists for the method.

D	In the JSON representation, the list of supported datums SHALL be provided as a dictionary (associative array) value for a `datums` property associating datum objects to the corresponding identifiers to be used as values for the `crs-datum` query parameter. These datum identifiers SHALL be safe CURIEs when a registered URI exists for the datum. The datum object SHALL include an `ellipsoid` property specifying the safe CURIE for the associated ellipsoid and may contain additional optional `title` and `description` properties.
E	In the JSON representation, the operation method objects SHALL include all valid parameters for that method as a dictionary (associative array) value for a `parameters` property to method parameters object (including optional `title` and `description` properties) to the corresponding identifiers to be used as values for the `crs-proj-params` query parameter. These method parameters SHALL be safe CURIEs when a registered URI exists for the parameter. To avoid repeating the same parameter, those objects may use a JSON pointer (`$ref`) to a top-level `parameters` property in the same custom projections JSON document.
F	In the JSON representation, the operation method objects SHALL include `centerLatParam` and/or `centerLonParam` properties (as applicable) whose values SHALL be the identifiers corresponding to the parameters for which values specified for the `crs-proj-center` query parameter will be mapped, in a manner consistent with requirement `/req/projection/crs-proj-center-definition`.

NOTE Refer to the OpenAPI definition for the `/projectionsAndDatums` path resource for the schema corresponding to these requirements.

Example 7. Example JSON response for `/projectionsAndDatums` resource listing available projections, parameters and datums


```
{
   "methods":
   {
      "[epsg-method:9802]":
      {
         "title": "Lambert Conic Conformal",
         "description": "Lambert Conic Conformal projection (2 standard
parallels)",
         "parameters":
         {
            "[epsg-parameter:8823]": { "$ref": "#/parameters/[epsg-
parameter:8823]" },
            "[epsg-parameter:8824]": { "$ref": "#/parameters/[epsg-
parameter:8824]" },
         },
         "centerLatParam": "[epsg-parameter:8823]"
      },
      "[epsg-method:9805]":
      {
         "title": "Mercator",
         "description": "Mercator projection",
         "parameters":
         {
            "[epsg-parameter:8823]": { "$ref": "#/parameters/[epsg-
parameter:8823]" },
            "[epsg-parameter:8802]": { "$ref": "#/parameters/[epsg-
parameter:8802]" }
         },
         "centerLatParam": "[epsg-parameter:8823]",
         "centerLonParam": "[epsg-parameter:8802]"
      },
      "[epsg-method:9807]":
      {
         "title": "Transverse Mercator",
         "description": "Transverse Mercator projection",
         "parameters":
         {
            "[epsg-parameter:8801]": { "$ref": "#/parameters/[epsg-
parameter:8801]" },
            "[epsg-parameter:8802]": { "$ref": "#/parameters/[epsg-
parameter:8802]" }
         },
         "centerLatParam": "[epsg-parameter:8801]",
         "centerLonParam": "[epsg-parameter:8802]"
      },
      "[epsg-method:9820]":
      {
```

```
         "title": "Lambert Azimuthal Equal Area",
         "description": "Lambert Azimuthal Equal Area projection",
         "parameters":
         {
            "epsg-parameter:8801": { "$ref": "#/parameters/[epsg-parameter:8801]"
},
            "epsg-parameter:8802": { "$ref": "#/parameters/[epsg-parameter:8802]"
}
         },
         "centerLatParam": "[epsg-parameter:8801]",
         "centerLonParam": "[epsg-parameter:8802]"
      },
      "[epsg-method:9829]":
      {
         "title": "Polar Stereographic",
         "description": "Polar Stereographic projection",
         "parameters":
         {
            "[epsg-parameter:8832]": { "$ref": "#/parameters/[epsg-
parameter:8832]" },
            "[epsg-parameter:8833]": { "$ref": "#/parameters/[epsg-
parameter:8833]" },
         },
         "centerLatParam": "[epsg-parameter:8832]",
         "centerLonParam": "[epsg-parameter:8833]"
      },
      "[epsg-method:9840]":
      {
         "title": "Orthographic",
         "description": "Orthographic projection",
         "parameters":
         {
            "epsg-parameter:8801": { "$ref": "#/parameters/[epsg-parameter:8801]"
},
            "epsg-parameter:8802": { "$ref": "#/parameters/[epsg-parameter:8802]"
}
         },
         "centerLatParam": "[epsg-parameter:8801]",
         "centerLonParam": "[epsg-parameter:8802]"
      },
      "mollweide":
      {
         "title": "Mollweide",
         "description": "Mollweide projection",
         "parameters":
         {
            "epsg-parameter:8802": { "$ref": "#/parameters/[epsg-parameter:8802]"
}
         },
         "centerLonParam": "[epsg-parameter:8802]"
      }
```

```
    },
    "parameters":
    {
       "[epsg-parameter:8801]": { "title": "Latitude of natural origin" },
       "[epsg-parameter:8802]": { "title": "Longitude of natural origin" },
       "[epsg-parameter:8823]": { "title": "Latitude of 1st standard parallel" },
       "[epsg-parameter:8824]": { "title": "Latitude of 2nd standard parallel" },
       "[epsg-parameter:8832]": { "title": "Latitude of standard parallel" },
       "[epsg-parameter:8833]": { "title": "Longitude of origin" }
    },
    "datums":
    {
       "[epsg-datum:6230]":
       {
          "title": "European Datum 1950",
          "description": "European Datum 1950",
          "ellipsoid": "[epsg-ellipsoid:7022]"
       },
       "[epsg-datum:6326]":
       {
          "title": "WGS84",
          "description": "World Geodetic System 1984 ensemble",
          "ellipsoid": "[epsg-ellipsoid:7030]"
       }
    }
}
```

Chapter 19. Requirement Class "Collection Map"

19.1. Overview

Requirements Class Collection Map	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/collection-map	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core
Dependency	https://www.opengis.net/spec/ogcapi-common-2/1.0/req/collections

The Collection Map requirements class specifies how to get maps from particular resources that contains geodata. Common resources that can contain geodata are the ones at the endpoint `/collections/{collectionId}` defined by *OGC API - Common* or *OGC API - Features* 1.0 Standard.

19.2. General

Recommendation 13	**/rec/collection-map/api-common**
A	Developers implementing the OGC API – Maps Standard should consider using the requirement classes specified in the https://www.opengis.net/spec/ogcapi-common-1/1.0/req/core (OGC API - Common version 1.0).

The Collection Maps requirements class depends on the *OGC API - Common - Part 2 - Geospatial data* "collections" requirement classes but remains flexible and does not require using *OGC API - Common - Part 1: Core*. This allows for other API architectures outside the OGC API framework to implement OGC API – Maps requirements classes. However, OGC API servers are normally expected to implement *OGC API - Common - Part 1*. If so, in practice, this means that the landing page and the conformance page follow *OGC API - Common - Part 1: Core*. It is also possible to combine this building block with *OGC API - Features* version 1.0 that is not tied to *OGC API - Common*, which can also be substituted for the *OGC API - Common - Part 2* dependency.

19.3. Geospatial data resources

The Maps API Standard does not specify how geospatial resources are exposed in the API or if they can be retrieved as raw geospatial data (e.g., feature items or coverage cell values). The Maps API Standard expects that other OGC API Standards will define how to expose additional access mechanisms for these geospatial resources. Examples of such standards, all of which also implement *OGC API - Common - Part 2: Geospatial data* defining the `/collections` and `/collections/{collectionId}` paths, are:

- *OGC API - Tiles* [https://docs.ogc.org/is/20-057/20-057.html] (Tiled vector data and tiled coverage data)
- *OGC API - Features* [https://docs.ogc.org/is/17-069r4/17-069r4.html] (feature collections and features)
- *OGC API - Coverages* [https://docs.ogc.org/DRAFTS/19-087.html] (coverages and data cubes)
- *OGC API - EDR* [https://docs.ogc.org/is/19-086r6/19-086r6.html] (retrieval of spatiotemporal data)
- *OGC API - Processes - Part 3: Workflows and Chaining* [https://docs.ogc.org/DRAFTS/21-009.html] ("Collection Output" requirements class)

The geospatial resources managed by these OGC API Standards (Tiled vector data and tiled coverage data, feature collections and features, coverages and data cubes, spatiotemporal data and collection outputs as a result of a process execution) can be modified or complemented by other resources creating new endpoints indirectly giving access to the modified or complemented resources. Other OGC API Standards will define these mechanisms. Examples of OGC API Standards doing that are:

- *OGC API - Styles* [https://docs.ogc.org/DRAFTS/20-009.html] defines how to complement a geospatial resource by adding a style to it. The new resource is an overlap of both resources.

NOTE	The concept of geospatial resource path substitutes the concept of "layer" in WMS but it intends to give a better integration between data visualization and data access.

Requirement 44	**/req/collection-map/desc-links**
A	If the API implementing the Maps API Standard has a mechanism for geospatial resources or modified geospatial resources to expose links to related aspects (e.g. feature items, metadata…), the API endpoint SHALL include a link with and with rel: `https://www.opengis.net/def/rel/ogc/1.0/map` and the href pointing to a the map resource that presents this geospatial data resource.

Permission 8	**/rec/collection-map/desc-links**
A	The collection can be exposed as a map totally or partially.

Requirement 45	**/req/collection-map/desc-crs**
A	The `crs` property in the collection object of a geospatial collection SHALL contain URI or safe CURIEs for the list of CRSs supported by the server for that collection.

B	If the collection is available more efficiently (e.g., if it is stored in the server in that CRS) using a particular CRS (the native CRS, also *called storage CRS*) that is not https://www.opengis.net/def/crs/OGC/1.3/CRS84, a `storageCrs` property in the collection object of a geospatial collection SHALL be the URI or the safe CURIE for that CRS.
C	If a `storageCrs` property is used and that is not https://www.opengis.net/def/crs/OGC/1.3/CRS84, an extent SHALL be provided in a `storageCrsExtent` property following the same schema as the extent property.

NOTE A CURIE `{authority}[-{objectType}]:{id}` would map to the following OGC URI: `https://www.opengis.net/def/{objectType}/{authority}/0/{id}`. If `-{objectType}` is missing, the default object type is *crs*.

Recommendation 14	**/rec/collection-map/storage-epoch**
A	If the collection is available more efficiently using a particular CRS (exposed in the `storageCrs` parameter) that is a dynamic coordinate reference system, the property `storageCrsCoordinateEpoch` in the collection object of the geospatial collection SHOULD provide the epoch of that CRS.

Example 8. Fragment of a collection with a links array with one item of the array pointing to a map.

```
  "id": "buildings",
  "title": "Buildings in the city of Bonn",
  "description": "This collection contains buildings",
  "attribution": "OpenStreetMap",
  "extent": {
    ...
  },
  "crs": ["[EPSG:32631]", "[EPSG:23031]", "[EPSG:4326]"],
  "storageCrs": "[EPSG:32631]",
  "storageCrsExtent": {
    "spatial" : {
         "bbox" : [ [ 47736, 4421022, 797736, 4734022 ] ]
    }
  },
  "storageCrsCoordinateEpoch": 2022.3,
  "links": [
    ...
    {
      "href": "https://data.example.com/collections/buildings/map",
      "rel": "https://www.opengis.net/def/rel/ogc/1.0/map",
      "type": "image/png",
    }
  ]
}
```

NOTE

In the WMS Standard, layers have a hierarchical dependency. At the time this version of the Maps API Standard was approved, neither *OGC API - Features* v1 nor *OGC API - Common* defined such a concept. However, there was a draft proposal to establish a hierarchical relation between collections based on a `parent` property and/or an `up` link relation type in the collection description. If a collection represented a hierarchy, then the Collections Selection requirements class could allow the selection of a subset of the collections it consists of to be included using the `collections` parameter.

19.4. Geospatial Data Resource Map

The core requirement class defines a map resource that is associated with an operation that contains the necessary information to later formulate a map request. Nevertheless, the Maps API core does not specify how to retrieve a map resource representing another resource. This section provides one possible workflow to support that use case from a geospatial data resource offered by an implementation of the Maps API.

19.4.1. Map path

Requirement 46	/req/collection-map/map-operation
A	Every geospatial data resource available as a map SHALL support a path URL and a HTTP GET operation to retrieve maps
B	The URI SHALL be composed by two parts: The initial is the URI of the geospatial data resource that can be represented as a map (e.g. /collections/{collectionId}) and the final part follows the pattern `/map`

This requirements class does not specify any additional query parameter for the GET request.

19.4.2. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

A successful response is a map that represents the collection of the geospatial data resource.

Chapter 20. Requirement Class "Dataset Map"

20.1. Overview

Requirements Class Dataset Map	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/dataset-map	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core https://www.opengis.net/spec/ogcapi-common-1/1.0/req/core https://www.opengis.net/spec/ogcapi-common-1/1.0/req/landing-page

The Maps API core requirements class defines how to retrieve a map from a resource but does not detail which resources can be "mapped". The Dataset Map requirements class defines how to retrieve a map resource from the dataset represented by the service. In other words, it provides the path to retrieve a map resource from the dataset behind the API endpoint. This is achieved by adding the map path to the root (a.k.a. the landing page) of the service.

20.2. General

The Dataset Map describes how to serve maps for a dataset provided as Web API instance implementing *OGC API - Common - Part 1* requirement classes or OGC API - Features - Part 1: Core, version 1.0 [https://www.opengis.net/doc/IS/ogcapi-features-1/1.0] as an alternative.

20.3. Dataset API landing page

The landing page provides links to start exploring the resources offered by the API endpoint. The landing page mainly consists of a list of links to root resources. The Maps API Standard extension specifies a new link in the landing page for getting a dataset map.

20.3.1. Response

The root path of an OGC API offering access to geospatial data represents a dataset that will be exposed in different ways by the API.

Sub-resources can be added to the root path of an OGC API endpoint to complement it by offering new endpoints indirectly giving access to the complemented dataset. Other OGC API Standards will define this mechanism. An example is OGC API - Styles which defines how to complement a geospatial resource by adding a style to it. The new resources is an overlap of the dataset and the resource.

Requirement 47	**/req/dataset-map/landingpage**

A	If a deployed API endpoint has a mechanism to expose root resources (e.g. a landing page), the API endpoint SHALL advertise a URI to retrieve dataset maps defined by this service as links to the descriptions paths with rel: `https://www.opengis.net/def/rel/ogc/1.0/map`.

In the landing page, in JSON format, the links follow the link schema defined in the OGC API - Common or in the OGC API - Features v1 Standards. Below you can find an example fragment of the response to an OGC API - Maps landing page showing the link to the dataset map.

Requirement 48	**/req/dataset-map/desc-extent**
A	An extent CRS SHALL be provided in an "extent" property following the same schema as the "extent" property for the collection (see OGC API - Common: Part 2).

NOTE The spatial bounding box is provided in https://www.opengis.net/def/crs/OGC/1.3/CRS84 for Earthly data.

Requirement 49	**/req/dataset-map/desc-crs**
A	The `crs` property in the landing page of a dataset SHALL contain a URI or a safe CURIE for the list of CRSs supported by the dataset as a whole.
B	If the dataset is available more efficiently using a particular CRS that is not https://www.opengis.net/def/crs/OGC/1.3/CRS84, a `storageCrs` property in the landing page of a dataset SHALL have the URI or the safe CURIE for that CRS as a value. For example, it may be more efficient to retrieve a map in the same CRS used to store the data. Note that the native CRS is also called the *storage CRS*.
C	If a `storageCrs` property is used and that is not https://www.opengis.net/def/crs/OGC/1.3/CRS84, an extent SHALL be provided in a `storageCrsExtent` property following the same schema as the extent property.

NOTE A CURIE `{authority}[-{objectType}]:{id}` would map to the following OGC URI: `https://www.opengis.net/def/{objectType}/{authority}/0/{id}`. If `-{objectType}` is missing, the default object type is *crs*.

Recommendation 15	**/rec/dataset/storage-epoch**

A	If the dataset is available more efficiently using a particular CRS (exposed in the `storageCrs` parameter) that is a dynamic coordinate reference system, the property `storageCrsCoordinateEpoch` in the landing page of the dataset SHOULD provide the epoch of that CRS.

Example 9. API Landing Page fragment that advertises the dataset map, the storageCrs and the extent of it.

```
{
  "title": "Dataset of the city of Bonn",
  "description": "Dataset containing collections such as buildings",
  "attribution": "OpenStreetMap",
  "extent": {
    "spatial" : {
        "bbox" : [ [ -2.2830363, 39.8188272, 6.6350523, 42.7010011 ] ]
    }
  },
  "crs": ["EPSG:32631", "EPSG:23031", "EPSG:4326"],
  "storageCrs": "EPSG:32631",
  "storageCrsExtent": {
    "spatial" : {
        "bbox" : [ [ 47736, 4421022, 797736, 4734022 ] ]
    }
  },
  "storageCrsCoordinateEpoch": 2022.3,
  "links": [
    ...,
    {
      "href": "https://data.example.org/map",
      "rel": "https://www.opengis.net/def/rel/ogc/1.0/map",
      "type": "image/png",
      "title": "Map combining collections in the dataset"
    }
  ]
}
```

20.4. Dataset maps

20.4.1. Map path

Requirement 50	**/req/dataset-map/operation**

A	If the dataset is exposed as a map in the default style, the Maps API implementation SHALL support a path URL and a HTTP GET operation to retrieve the map the API endpoint provides. The URI SHALL be `/map`

The operation request does not define extra parameters. Servers can advertise the requirements class https://www.opengis.net/spec/ogcapi-tiles-1/1.0/req/collections-selection to add a `collections` parameter that can be used to include only parts of the dataset in the map representation.

20.4.2. Response

A successful GET response is described in the core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

A successful response is a map resource that represents the entire dataset. In a Web API providing access to a complex dataset comprised of several geospatial data resources, there are reasons for being selective in the amount of information included. This can be achieved by applying a manual filter described in the Requirement Class "Collection Selection" or as an automatic decision by the server side.

Recommendation 16	**/rec/dataset-map/geodata-selection**
A	When it is possible and sensible, all geospatial data resources (`/collections`) supporting the requested CRS SHOULD be represented in the map response.

Permission 9	**/per/dataset-map/geodata-selection**
A	If it is not possible and sensible to represent all geospatial data resources (`/collections`) in a map (e.g. it compromises performance or maps are become packed with too many elements), the server is allowed to select only the most significant geospatial data resources.

Chapter 21. Requirement Class "Styled Maps"

21.1. Overview

Requirements Class Styles Map	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/styled-map	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

The Styles Map requirements class specifies how to get maps from particular resources with a style applied by *OGC API - Styles* 1.0.

NOTE This mechanism replaces the parameter `STYLES` in WMS 1.3 and adds the possibility to have a generic style for all layers of the service at once.

21.2. Styled resources

Geospatial resources can be modified or complemented by styles creating new endpoints giving access to the modified or complemented resources. The way this is done to a dataset resource or to a collection is specified in the draft *OGC API – Styles* (as of December 2023).

Requirement 51	**/req/styled-map/desc-links**
A	If the deployed API endpoint has a mechanism for styled datasets or geospatial resources to expose links to resources associated to the style, the API endpoint SHALL include a link with and with rel: `https://www.opengis.net/def/rel/ogc/1.0/map` and the href pointing to a the map resource that presents this styled resource.

Example 10. Fragment of a collection with a links array with one item of the array pointing to a map.

```
  "id": "buildings",
  "title": "Buildings in the city of Bonn",
  "description": "This collection contains buildings",
  "attribution": "OpenStreetMap",
  "extent": {
    ...
  }
  "links": [
    ...
    {
      "href": "https://data.example.com/collections/buildings/styles/dark/map",
      "rel": "https://www.opengis.net/def/rel/ogc/1.0/map",
      "type": "image/png",
    }
  ]
}
```

21.3. Styled Resource Map

Styled resources are defined by the draft OGC API – Styles Standard (as of December 2023). Nevertheless, that draft does not specify how to retrieve a map resource representing a styled resource. This section explains how to do this.

21.3.1. Map path

Requirement 52	**/req/styled-map/map-operation**
A	Every styled resource available as a map SHALL support a path URL and a HTTP GET operation to retrieve maps
B	The URI SHALL be composed by two parts: The initial part is the URI of the styled resource that can be represented as a map (e.g., /collections/{collectionId}/styles/{styleId}) and the final part follows the pattern `/map`

The Maps API Standard does not specify any additional query parameter in the GET request.

21.3.2. Response

A successful GET response is described in the Maps API core class (https://www.opengis.net/spec/ogcapi-maps-1/1.0/conf/core).

A successful response is a map that represents the styled resource.

Chapter 22. Requirements classes for encodings

The *OGC API - Maps* Standard is designed to support mapped data that can potentially be encoded and provided in existing geospatial formats or new formats that could be developed in the future. Web API deployments may adopt these encodings and declare conformance to them in the list of conformance classes supported by the Web API. The requirements in this section do not limit the number of encodings offered by a Maps Web API deployment. The intent is to provide a minimum set of encodings that could be implemented by any deployed Maps APIs as well as to provide a practical way to test conformance to this Standard. For each of these encodings, a requirements class is defined. A Web API implementing OGC API – Maps classes are free to support other encodings and data formats. These other formats may be more convenient for given use cases even though they may not be listed as supported conformance classes by the Web API deployment. In addition, the declaration of an encoding in the conformance classes supported does not mean that all the resources provided by an OGC enabled API endpoint should support all of them. Partial support could be conditioned by the nature of the data behind each collection.

22.1. Overview

Six requirements classes for encodings map responses of *OGC API - Maps* implementation are defined:

- PNG
- JPEG
- JPEG XL
- TIFF
- SVG
- HTML

NOTE	If maps are provided as tiles, this section should be ignored and the equivalent section in the *OGC API – Tiles Tiles* Standard should be used instead.

NOTE	None of the encodings specified here are mandatory and an implementation of the Maps API standard may implement none of them but implement other encodings instead.

22.2. Requirement Class "PNG"

PNG can be used for simple and immediate visualization in a web browser. For this use case, selecting an encoding that can be natively interpreted by the web browser is fundamental. The PNG format is one of the most popular formats. The PNG format is defined by the ISO Information technology Computer graphics and image processing Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E), also available by the W3C in https://www.w3.org/TR/PNG.

PNG supports lossless data compression. PNG supports palette-based images (with palettes of 24-bit RGB or 32-bit RGBA colors), grayscale images (with or without an alpha channel for transparency), and full-color non-palette-based RGB or RGBA images. The PNG working group designed the format for transferring images on the Internet, not for professional-quality print graphics. Therefore, non-RGB color spaces such as CMYK are not supported.

Requirements Class PNG	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/png	
Target type	Web API
Dependency	[ISO/IEC 15948 standard]
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

Requirement 53	**/req/png/content**
A	Every 200-response of the server with the media type image/png SHALL be a PNG document representing only one map.
B	The colors of the PNG SHALL represent the geospatial features or coverage values in the map.
C	The alpha channel of the PNG SHALL be used when partial transparency is required
D	All maps representing parts of the same resource or resources and using the same style SHALL follow the same portrayal rules

NOTE The way the colors in the PNG format are mapped to geospatial features or coverage values is out of scope for the Maps API Standard. However, a common set of portrayal rules for all maps representing part of the same resource is essential, as maps are normally represented one after the other as a response to actions of the user (e.g. zoom and pan).

22.3. Requirement Class "JPEG"

JPEG can be used for simple and immediate visualization in a web browser. In these circumstances, selecting an encoding that can be natively interpreted by the web browser is fundamental. The JPEG format is another popular format used in web applications. JPEG is defined by the ITU-T Recommendation T.81 and the ISO/IEC 10918-1.

The JPEG compression algorithm operates at its best on photographs and paintings with smooth variations of tone and color. It is best used for color and grayscale still images, but not for binary images. JPEG is also the most common format used by digital cameras. However, JPEG is not well suited for line drawings and other textual or iconic graphics, where the sharp contrasts between adjacent pixels can cause noticeable artifacts. Such images are better saved in a lossless graphics

format such as PNG. Because JPEG uses a lossy compression method, which reduces image fidelity. As such, it is inappropriate for exact reproduction of imaging data.

Requirements Class JPEG	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/jpeg	
Target type	Web API
Dependency	[ISO/IEC 10918-1]
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

Requirement 54	**/req/jpeg/content**
A	Every 200-response of the server with the media type image/jpeg SHALL be a JPEG document representing only one map.
B	The colors of the JPEG SHALL represent geospatial features and/or coverage values in the map.
C	All maps representing parts of the same resource or resources and using the same style SHALL follow the same portrayal rules.

NOTE The way the colors in the JPEG are mapped to geospatial features or coverage values is out of scope of the Maps API Standard. However, a common set of portrayal rules for all maps representing part of the same resource is essential, as maps are normally represented one after the other as a response to actions of the user (e.g., zoom and pan).

NOTE JPEG is an ideal support for remote sensing imagery. The use of JPEG to represent linear features or color solid polygons is not recommended.

22.4. Requirement Class "JPEG XL"

JPEG XL [https://jpeg.org/jpegxl/] is a relatively newer image format (compared to PNG and JPEG) with support for both lossy and lossless compression, alpha channels, high bit depths, high image quality, high compression ratio as well as high performance encoding and decoding. The format and codec are unencumbered of any patent. Free and open source libraries are available to read and write the format such as libjxl [https://github.com/libjxl/libjxl], released under a 3-clause BSD license. Although JPEG XL is not yet widely supported in web browsers at the time of writing this document, this situation is likely to change, and this requirement class is included due to the numerous advantages of JPEG XL over the traditional JPEG and PNG encoding, such as avoiding the need for compromising between high compression ratio, image quality and translucency. JPEG XL is defined by the multi-part ISO standard Information technology JPEG XL - image coding system, including ISO/IEC 18181-1:2022 Part 1: Core coding system [https://www.iso.org/standard/77977.html] and ISO/IEC 18181-2:2021 Part 2: File format [https://www.iso.org/standard/80617.html].

Requirements Class JPEGXL	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/jpegxl	
Target type	Web API
Dependency	[JPEG_XL1]
Dependency	[JPEG_XL2]
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

Requirement 55	/req/jpegxl/content
A	Every 200-response of the server with the media type image/jxl SHALL be a JPEG XL file representing only one map.
B	The JPEG XL SHALL be a color image representing the geospatial features or coverage values in the map.
C	All maps representing parts of the same resource or resources and using the same style SHALL follow the same portrayal rules.

NOTE The way the colors in the JPEG XL are mapped to geospatial features or coverage values is out of scope of the Maps API Standard. However, a common set of portrayal rules for all maps representing part of the same resource is essential, as maps are normally represented one after the other as a response to actions of the user (e.g., zoom and pan).

NOTE JPEG XL is an ideal file format for all map types, due to its support for high image quality (including the possibility of lossless encoding), high compression ratio, high bit depth and translucency.

22.5. Requirement Class "TIFF"

One use case for maps is to distribute coverage values as regular grids. In these circumstances, selecting an encoding able to store grid values in their original format and eventually compress them using lossless compression is the right solution. The TIFF format is one of the older formats but is still one of the most popular formats to preserve arrays of data values and defined by the Adobe Systems TIFF v6 specification.

TIFF is a flexible, adaptable file format for handling images and data. The ability to store data in a lossless format makes a TIFF file a useful image archive: Unlike standard JPEG files, a TIFF file using lossless compression such as PackBits or LZW compression.

Requirements Class TIFF
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/tiff

Target type	Web API
Dependency	[TIFF_V6]
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

Requirement 56	**/req/tiff/content**
A	Every 200-response of the server with the media type image/tiff SHALL be a TIFF document representing only one map.
B	The TIFF file SHALL represent colors by using an image palette or RGB combination.
C	All maps representing parts of the same resource or resources and using the same style SHALL follow the same portrayal rules or represent data with the same reference and units of measure.

NOTE	TIFF is an ideal support for geospatial grid data values in its original format. This is not allowed for OGC API - Maps and it is expected that OGC API endpoints dealing with coverages will support this mode.

Recommendation 17	**/rec/tiff/geotiff**
A	A TIFF encoding SHOULD include georeference information using the OGC GeoTIFF Standard [https://docs.ogc.org/is/19-008r4/19-008r4.html].

22.6. Requirement Class "SVG"

SVG is a commonly used format for representing vector objects on the web. SVG is simple to understand and well supported by tools and software libraries. Modern web browsers can render the SVG format directly and can also react to events generated by the user on the features visualized. That is why it is a good alternative for creating more interactive maps.

Requirements Class SVG	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/svg	
Target type	Web API
Dependency	[SVG]
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core

Requirement 57	**/req/svg/content**

A	Every 200-response of the server with the media type image/svg+xml SHALL be a SVG document representing only a map.
B	The SVG coordinates inside the map SHALL start at 0,0 and end in the width and height of the request.

Permission 10	**/rec/svg/overflow**
A	A SVG content of a map can contain features that are partially outside of the requested map bounding box (represented by negative coordinates or coordinates bigger than the requested width and height).

22.7. Requirement Class "HTML"

Returning a HTML page with a full, operational map browser that allows for interactive navigation over the map (e.g., zoom and pan) is convenient for easy exploration of the data or for testing purposes.

Requirements Class HTML	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/html	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/core
Dependency	https://html.spec.whatwg.org/: HTML Living Standard

Requirement 58	**/req/png/content**
A	Every 200-response of the server with the media type text/html SHALL be a HTML document representing the geospatial data as maps.

NOTE	An HTML page can contain links to other resources. In particular, it can contain image sources linked to map resources as image/png or image/jpeg that will represent the actual data.

NOTE	An HTML page may contain MapML elements to represent the map (https://maps4html.org/MapML/spec/).

Recommendation 18	**/rec/html/content**

A	A HTML encoding MAY internally use requests to maps in image/png or image/jpeg format using this OGC API, to present the map images (e.g. in img src tags) to start and as a response of the actions of the user (e.g. zoom and pan).

Chapter 23. Requirements Class "API Operations"

23.1. Overview

Requirements Class API Operations	
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/api-operations	
Target type	Web API
Dependency	https://www.opengis.net/spec/ogcapi-common-1/1.0/req/landing-page

The "API Operations" requirements class defines requirements for providing a definition of a Web API implementing the Maps API Standard using an API definition language. This requirement class is intended to be combined with another requirement class specifying requirements for providing an API definition in a particular API definition language and / or version, such as the Open API 3.0 requirement class defined in OGC API - Common - Part 1 :Core [https://docs.ogc.org/is/19-072/19-072.html#_ab6e3c2d-d2dc-4f01-a7d4-8b52133289a0], or another eventual requirement class for OpenAPI 3.1.

23.2. Web API description

The API definition provides a description of the complete list of API resources available at an endpoint. Reading this description, an application would have the full picture of the resources that the API implementation provides, how to retrieve resources, and what responses are expected for successful and unsuccessful requests. Without an API description or documentation, an application would be forced to traverse all links, starting with the landing page, to get an equivalent full list of resources.

The *OpenAPI Specification 3.0 (oas30)* requirement class from *OGC API - Common - Part 1: Core* provides many details on general requirements that can be used in conjunction with this requirements class. A later version of OGC API - Common may describe how to declare support for other versions of OpenAPI. This requirement class depends on the *OGC API - Common* landing page requirements class which provides details on how to request an API definition.

23.2.1. Response

Completeness

The API definition resulting as a response to this request need to take into consideration the relevant resources specified in the Maps API Standard.

Requirement 59	/req/api-operations/completeness

A	The API definition SHALL provide paths for all map, custom projections, tileset, tilesets list and tile resources provided by the API instance.
B	The resource paths defined in the API definition SHALL be consistent with the links to the same resources provided by the landing page, collections, tileset and tilesets list resources.
C	The resource paths defined in the API definition SHALL provide the description of the parameters that the map, tileset and tile resources need to operate that are specified in corresponding conformance classes.

Reusable API components

Reusable components for creating API definitions for implementations of this OGC API - Maps using OpenAPI 3.0 can be found in https://schemas.opengis.net/ogcapi/maps/part1/1.0/openapi .

NOTE	While this specification is in draft status, the components are available from https://github.com/opengeospatial/ogcapi-maps/tree/master/openapi .

A server implementation of the *OGC API – Maps Standard* can use the content in the openapi directory to generate a response for the API description using OpenAPI 3.0. The *ogcapi-maps-1.yaml* includes paths and components. An implementation should only include the paths that are implemented and remove the references to the paths that are not implemented. The components part includes parameters, responses and schemas that can be reused as-is. The `api` directory contains JSON files that are templates with enumerated values for collections, styles and tile matrix sets. A particular implementation of the Maps API should enumerate the actual resources exposed by the API deployment in the same way. The server can select to dynamically implement responses to /api/* (where * is replaced by all-collections, styles,…) or hardcode the /api/* files with the actual list of resource identifiers in the enumerations.

To improve performance, the whole content of this directory can be bundled into a single document by executing a tool such as *swagger-cli*. This document can be served for the *OGC API - Common - Part 1* `service-desc` link from the landing page.

Path Operation Identifiers

The API definition provides a client application with a set of paths that the client can use to interact with the API endpoint and get new resources. The API description of each path provides a description of what parameters to use in the request and what to expect in the response. However, the Maps API Standard does not propose a fixed set of paths so there is an issue identifying the requirements classes pertaining to each path in an API instance. In other words, the API description alone does not provide enough information by itself. Therefore, there is a need to associate the resource paths in the API definition with the resources defined in the various requirements classes. The Maps APIs Standard proposes a suffix mechanism to be applied to the operation identifiers of the path (e.g., the `operationId` property in OpenAPI Specification 3.0) to list the requirement classes

pertaining to each path. Each path should have a unique operationId suffix, so it is expected that the API instance provides a prefix to the proposed suffixes that make each operation identifier unique.

Requirement 60	**/req/api-definition/operation-id**
A	For API definitions with a concept of operation identifiers, the paths defined in the API definition SHALL have an operation identifier value ending with the relevant dot-separated suffix corresponding to the resource as specified in Table 10.

Table 12. API operation identifier suffixes

Origin	**Styled**	**Resource**	**Operation id suffixes**
root		Projections	`getCustomCRSProjections`
With the origins described in this document			
DataSet[5]		Map[1]	`.dataset.getMap`
DataSet[5]		TileSetsList[4]	`.dataset.map.getTileSetsList`
DataSet[5]		TileSet[3]	`.dataset.map.getTileSet`
DataSet[5]		Tile[2]	`.dataset.map.getTile`
DataSet[5]	Styled[7]	Map[1]	`.dataset.style.getMap`
DataSet[5]	Styled[7]	TileSetsList[4]	`.dataset.style.map.getTileSetsList`
DataSet[5]	Styled[7]	TileSet[3]	`.dataset.style.map.getTileSet`
DataSet[5]	Styled[7]	Tile[2]	`.dataset.style.map.getTile`
Collection[6]		Map[1]	`.collection.getMap`
Collection[6]		TileSetsList[4]	`.collection.map.getTileSetsList`
Collection[6]		TileSet[3]	`.collection.map.getTileSet`
Collection[6]		Tile[2]	`.collection.map.getTile`
Collection[6]	Styled[7]	Map[1]	`.collection.style.getMap`
Collection[6]	Styled[7]	TileSetsList[4]	`.collection.style.map.getTileSetsList`
Collection[6]	Styled[7]	TileSet[3]	`.collection.style.map.getTileSet`
Collection[6]	Styled[7]	Tile[2]	`.collection.style.map.getTile`
With other potential origins[8]			
other		Map[1]	`#.getMap`
other		TileSetsList[4]	`#.map.getTileSetsList`
other		TileSet[3]	`#.map.getTileSet`
other		Tile[2]	`#.map.getTile`
other	Styled[7]	Map[1]	`#.style.getMap`

Origin	Styled	Resource	Operation id suffixes
other	Styled[7]	TileSetsList[4]	`#.style.map.getTileSetsList`
other	Styled[7]	TileSet[3]	`#.style.map.getTileSet`
other	Styled[7]	Tile[2]	`#.style.map.getTile`

[1] The *Map* resource is defined in requirements class "Core".

[2] The *Tile* resource is defined in the *OGC API - Tiles - Part 1: Core* "Core" requirements class.

[3] The *TileSet* resource is defined in the *OGC API - Tiles - Part 1: Core* "TileSet" requirements class.

[4] The *TileSetsList* resource is defined in the *OGC API - Tiles - Part 1: Core* "TileSets List" requirements class. Map tilesets are defined in the *Map Tilesets* requirements class_ and depend on *OGC API - Tiles - Part 1: Core*.

[5] The *DataSet* origin is defined in requirements class "Dataset Maps" and depends on *OGC API - Common - Part 1: Core*.

[6] The *Collection* origin is defined in requirements class "Collection Maps" and depends on the *Collections* requirements class defined in *OGC API - Common - Part 2: Geospatial data*.

[7] Styled tilesets rely on the ability to list styles defined in *OGC API - Styles*.

[8] '#' represents an optional *origin* that could be defined in another relevant standard.

Chapter 24. Requirements Class "CORS"

24.1. Overview

Requirements Class CORS	
Target type	Web API
https://www.opengis.net/spec/ogcapi-maps-1/1.0/req/cors	
Dependency	https://www.w3.org/TR/2020/SPSD-cors-20200602/

Many current implementations of map clients are done in HTML and JavaScript on HTTPS. A common situation is that a single JavaScript client combines maps from different Web APIs and domains. In this case, the Cross-Origin Resource Sharing (CORS) is applied by web browsers preventing access of data from a domain that is different from the Web API, if this is not explicitly allowed by the server headers. The CORS requirements class requires the implementation of CORS support for the Web APIs.

24.2. CORS for javascript clients

Requirement 61	/req/cors/cors
A	An implementation of OGC API - Maps that wants to allow JavaScript clients (e.g. Web Browser applications) from a domain different from the OGC API - Maps endpoint SHALL implement CORS as defined by W3C (https://www.w3.org/TR/2020/SPSD-cors-20200602/).

NOTE	It is not the purpose of this Standard to specify how CORS is implemented and that may change as web browsers and servers become more concerned about security (for example, currently a PNG in a <img src=""> is not affected by CORS but a PNG retrieved by the JavaScript fetch() function is affected). It is worth notice that a server that wants to authorize a JavaScript client application from another domain (under the CORS rules) should use the Access-Control-Allow-Origin: header to list which domains are authorized to read the the map (or use * to indicate that all domains are authorized). Services that want to comply to the open data sharing principles should be willing to allow all clients reading the data even if they are in other domains. However, this is up to the server implementers.

Annex A: Conformance Class Abstract Test Suite (Normative)

A.1. Conformance Class "Core"

A.1.1. Requirement Map Operation

Test id:	/conf/core/map-op
Requirement:	/req/core/map-op
Test purpose:	Verify that the implementation supports the map retrieval operation
Test method:	**Given:** a geospatial data resource conforming to the Maps API Standard, with an API path including `⋯/map⋯` or discovered following a link with relation type `[ogc-rel:map]` **When:** performing a GET operation on the `/map` resource with a supported media type specified in the `Accept:` header (e.g., `Accept: image/png,image/jpeg`) **Then:** - assert that the response has HTTP status code 200 - assert that the CRS of the resulting map is the native CRS (the native CRS is `[ogc:crs84]`, unless specified in the dataset map or collection map description `storageCrs` property, or specified out-of-band if *OGC API - Common* is not implemented), - assert that the response headers either includes a `Content-Crs:` header for the native CRS, or omits it if the nativeCRS is `[ogc:crs84]`, - assert that the headers of the response include a `Content-Bbox:` header with the actual axis-aligned geospatial boundary of the rendered map, - assert that the `Content-Bbox:` coordinates are in the native CRS and honor the CRS coordinates order, - assert that the body of the response is a map of the geospatial data resource in the negotiated format.

A.1.2. Requirement Map Response

Test id:	/conf/core/map-response
Requirement:	/req/core/map-response
Test purpose:	Verify that the implementation's response for the map retrieval operation is correct
Test method:	**Given:** a map resource that was successfully retrieved for /conf/core/map-op **When:** retrieving that resource for the map operation **Then:** (assert that all parts of the requirement are fully satisfied)

A.1.3. Requirement Map Conformance Success

Test id:	/conf/core/conformance-success
Requirement:	/req/core/conformance-success
Test purpose:	For implementations having a mechanism to advertise conformance classes, verify that it reports conformance to this Standard correctly.
Test method:	**Given:** a conformance resource in a recognized format, such as the OGC API JSON `/conformance` resource **When:** retrieving that resource from the API endpoint+ **Then:** (assert that all parts of the requirement are fully satisfied)

A.2. Conformance Class "Map Tilesets"

A.2.1. Requirement desc-links

Test id:	/conf/tilesets/desc-links
Requirement:	/req/tilesets/desc-links
Test purpose:	Verify that the implementation supports map tilesets
Test method:	**Given:** a geospatial data resource conforming to this Standard, to "Map Tilesets", to OGC *API - Tiles* and providing a description resource including links **When:** retrieving the geospatial data resource description **Then:** - assert that the geospatial data resource (e.g., collection or landing page description's `links` property) includes a link with the `href` pointing to a tileset list supported that presents a tile aspect of this geospatial data resource and with rel: `[ogc-rel:tilesets-map]`

A.2.2. Requirement tiles-parameters

Test id:	/conf/tilesets/desc-links
Requirement:	/req/tilesets/desc-links
Test purpose:	Verify that the implementation supports relevant parameters for map tilesets
Test method:	**Given:** a geospatial data resource conforming to this Standard, to "Map Tilesets", to OGC *API - Tiles*, and to *Maps* requirements classes introducing parameters relevant for map tiles **When:** retrieving the map tiles with parameters for the *background*, *display resolution*, *spatial subsetting* (only for `subset` and `subset-crs` parameters, and only if a vertical dimension is available), *general subsetting*, and *scaling* requirements classes **Then:** - assert that tiles responses reflect the relevant map parameters used for the requests

NOTE	This conformance class depends on *OGC API - Tiles - Part 1: Core* "Tilesets List" conformance class to which the implementation must also conform.

A.3. Conformance Class "Background"

A.3.1. Requirement `bgcolor` parameter definition

Test id:	/conf/background/bgcolor-definition
Requirement:	/req/background/bgcolor-definition
Test purpose:	Verify that the implementation supports the `bgcolor` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving a map without `bgcolor` parameter, with `bgcolor` using a hexadecimal value and with `bgcolor` using a W3C Web Color name **Then:** (assert that all parts of the requirement are fully satisfied)

A.3.2. Requirement `transparent` parameter definition

Test id:	/conf/background/transparent-definition
Requirement:	/req/background/transparent-definition
Test purpose:	Verify that the implementation supports the `transparent` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving a map for all combinations of (no `transparent` parameter, transparent=false `, `transparent=true`) and with and without `bgcolor` parameter **Then:** (assert that all parts of the requirement are fully satisfied)

A.3.3. Requirement Background Map Success

Test id:	/conf/background/map-success
Requirement:	/req/background/map-success
Test purpose:	Verify that the implementation's response for the map retrieval operation with a background color and/or transparent parameter is correct
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** for all combinations of (no `transparent` parameter, transparent=false `, `transparent=true`) and (without `bgcolor` parameter, with `bgcolor` using a hexadecimal value and with `bgcolor` using a W3C Web Color name) **Then:** (assert that all parts of the requirement are fully satisfied)

A.4. Conformance Class "Collection Selection"

A.4.1. Requirement `collections` parameter definition

Test id:	/conf/collections-selection/collections-parameter

Requirement:	/req/collections-selection/collections-parameter
Test purpose:	Verify that the implementation supports the `collections` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core and that is understood to consist of multiple collections (e.g., a dataset advertising support for Dataset Map and featuring multiple collections) **When:** retrieving a map using the `collections` parameter with one and multiple *collectionsIds* **Then:** (assert that all parts of the requirement are fully satisfied)

A.4.2. Requirement Collection Selection Response

Test id:	/conf/collections-selection/collections-response
Requirement:	/req/collections-selection/collections-response
Test purpose:	Verify that the implementation responds correctly to map requests using the `collections` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core and that is understood to consist of multiple collections (e.g., a dataset advertising support for Dataset Map and featuring multiple collections) **When:** retrieving a map using the `collections` parameter with one and multiple *collectionsIds* **Then:** (assert that all parts of the requirement are fully satisfied)

A.5. Conformance Class "Scaling"

A.5.1. Requirement `width` parameter definition

Test id:	/conf/scaling/width-definition
Requirement:	/req/scaling/width-definition
Test purpose:	Verify that the implementation supports the (scaling) `width` parameter correctly for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using `width` parameter for different values, as well as the same `bbox` parameter if spatial subsetting is supported, with and without `height` parameter, with and without `mm-per-pixel` parameter if display resolution is supported **Then:** (assert that all parts of the requirement are fully satisfied)

A.5.2. Requirement `height` parameter definition

Test id:	/conf/scaling/height-definition
Requirement:	/req/scaling/height-definition

Test purpose:	Verify that the implementation supports responds the (scaling) `height` parameter correctly for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using `height` parameter for different values, as well as the same `bbox` parameter if spatial subsetting is supported, with and without `width` parameter, with and without `mm-per-pixel` parameter if display resolution is supported **Then:** (assert that all parts of the requirement are fully satisfied)

A.5.3. Requirement `scale-denominator` parameter definition

Test id:	/conf/scaling/scale-denominator-definition
Requirement:	/req/scaling/scale-denominator-definition
Test purpose:	Verify that the implementation supports the `scale-denominator` parameter correctly for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `scale-denominator` parameter, combining all possibilities of with and without `width` and/or `height` parameters, with and without `bbox` and `center` parameter if spatial subsetting is supported, with and without `mm-per-pixel` parameter if display resolution is supported **Then:** (assert that all parts of the requirement are fully satisfied)

A.6. Conformance Class "Display Resolution"

A.6.1. Requirement `mm-per-pixel` parameter definition

Test id:	/conf/display-resolution/mm-per-pixel-definition
Requirement:	/req/display-resolution/mm-per-pixel-definition
Test purpose:	Verify that the implementation supports the `mm-per-pixel` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `mm-per-pixel` parameter, for different styles if styled maps are supported, combining all possibilities of with and without `width` and/or `height` parameters, with and without `bbox` and `center` parameter if spatial subsetting is supported, with and without `mm-per-pixel` parameter if display resolution is supported **Then:** (assert that all parts of the requirement are fully satisfied)

A.6.2. Requirement Display Resolution Map Success

Test id:	/conf/display-resolution/map-success
Requirement:	/req/display-resolution/map-success

Test purpose:	Verify that the implementation responds correctly to map requests using the `mm-per-pixel` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `mm-per-pixel` parameter, for different styles if styled maps are supported, combining all possibilities of with and without `width` and/or `height` parameters, with and without `bbox` and `center` parameter if spatial subsetting is supported, with and without `mm-per-pixel` parameter if display resolution is supported **Then:** (assert that all parts of the requirement are fully satisfied)

A.7. Conformance Class "Spatial Subsetting"

A.7.1. Requirement `bbox-crs` parameter definition

Test id:	/conf/spatial-subsetting/bbox-crs
Requirement:	/req/spatial-subsetting/bbox-crs
Test purpose:	Verify that the implementation supports the `bbox-crs` parameter for specifying the CRS of the `bbox` parameter correctly
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using `bbox` and `bbox-crs` parameter for different values, as well as different values for the `crs` parameter if supported and applicable, **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.2. Requirement `subset-crs` parameter definition

Test id:	/conf/spatial-subsetting/subset-crs
Requirement:	/req/spatial-subsetting/subset-crs
Test purpose:	Verify that the implementation supports the `subset-crs` parameter for specifying the CRS of the `subset` parameter correctly
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using `subset` and `subset-crs` parameter for different values (using the correct spatial axes), as well as different values for the `crs` parameter if supported and applicable, **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.3. Requirement `center-crs` parameter definition

Test id:	/conf/spatial-subsetting/center-crs
Requirement:	/req/spatial-subsetting/center-crs
Test purpose:	Verify that the implementation supports the `center-crs` parameter for specifying the CRS of the `center` parameter correctly

Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using `center` and `center-crs` parameter for different values, as well as different values for the `crs` parameter if supported and applicable, **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.4. Requirement `bbox` parameter definition

Test id:	/conf/spatial-subsetting/bbox-definition
Requirement:	/req/spatial-subsetting/bbox-definition
Test purpose:	Verify that the implementation supports the `bbox` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `bbox` parameter (with and without the `bbox-crs` parameter), **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.5. Requirement spatial subsetting `subset` parameter definition

Test id:	/conf/spatial-subsetting/subset-definition
Requirement:	/req/spatial-subsetting/subset-definition
Test purpose:	Verify that the implementation supports the `subset` parameter for spatial subsetting
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `subset` parameter (with and without the `subset-crs` parameter, for the correct spatial axes), **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.6. Requirement map subset response

Test id:	/conf/spatial-subsetting/subset-response
Requirement:	/req/spatial-subsetting/subset-response
Test purpose:	Verify that the implementation responds correctly to map requests using the `subset` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `subset` (with and without the `subset-crs` parameter) **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.7. Requirement `center` parameter definition

Test id:	/conf/spatial-subsetting/center-definition
Requirement:	/req/spatial-subsetting/center-definition

Test purpose:	Verify that the implementation supports the `center` parameter correctly
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `center` parameter (with and without the `center-crs` parameter), **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.8. Requirement subsetting `width` and `height` parameters definition

Test id:	/conf/spatial-subsetting/width-height
Requirement:	/req/spatial-subsetting/width-height
Test purpose:	Verify that the implementation supports the `width` and `height` parameter for spatial subsetting when used together with the `center` and/or the `scale-denominator` parameters
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `center` parameter together, with the `width` and/or `height` (with and without the `center-crs` parameter), with and without the `scale-denominator` parameter if scaling is supported **Then:** (assert that all parts of the requirement are fully satisfied)

A.7.9. Requirement map subset success

Test id:	/conf/spatial-subsetting/map-success
Requirement:	/req/spatial-subsetting/map-success
Test purpose:	Verify that the implementation responds correctly to map requests using subsetting parameters (`bbox`, `subset` or `center`)
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `bbox` (with and without the `bbox-crs` parameter), `subset` (with and without the `subset-crs` parameter), and `center` parameter (with and without the `center-crs` parameter, with the `width` and/or `height` parameter, with and without the `scale-denominator` parameter if scaling is supported **Then:** (assert that all parts of the requirement are fully satisfied)

A.8. Conformance Class "Date and Time"

A.8.1. Requirement `datetime` parameter definition

Test id:	/conf/datetime/datetime-definition
Requirement:	/req/datetime/datetime-definition
Test purpose:	Verify that the implementation supports the `datetime` parameter

Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `datetime` parameter **Then:** (assert that all parts of the requirement are fully satisfied)

A.8.2. Requirement `datetime` parameter response

Test id:	/conf/datetime/datetime-response
Requirement:	/req/datetime/datetime-response
Test purpose:	Verify that the implementation responds correctly to map requests using the `datetime` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `datetime` parameter **Then:** (assert that all parts of the requirement are fully satisfied)

A.8.3. Requirement temporal `subset` parameter definition

Test id:	/conf/datetime/subset-definition
Requirement:	/req/datetime/subset-definition
Test purpose:	Verify that the implementation supports temporal subsetting using the `subset` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `subset` parameter with the `time` axis **Then:** (assert that all parts of the requirement are fully satisfied)

A.8.4. Requirement temporal subset response

Test id:	/conf/datetime/subset-response
Requirement:	/req/datetime/subset-response
Test purpose:	Verify that the implementation responds correctly to temporal subsetting requests using the `subset` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `subset` parameter with the `time` axis **Then:** (assert that all parts of the requirement are fully satisfied)

A.8.5. Requirement temporal axis

Test id:	/conf/datetime/axis
Requirement:	/req/datetime/axis
Test purpose:	Verify that the implementation supports the `time` axis for temporal subsetting using the `subset` parameter

Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `subset` parameter with the `time` axis **Then:** (assert that all parts of the requirement are fully satisfied)

A.8.6. Requirement temporal subsetting success

Test id:	/conf/datetime/map-success
Requirement:	/req/datetime/map-success
Test purpose:	Verify that the implementation responds correctly to temporal subsetting requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `subset` parameter with the `time` axis **Then:** (assert that all parts of the requirement are fully satisfied)

A.9. Conformance Class "General Subsetting"

A.9.1. Requirement uniform additional dimensions

Test id:	/conf/general-subsetting/uniform-additional-dimensions
Requirement:	/req/general-subsetting/uniform-additional-dimensions
Test purpose:	Verify that the implementation describes the extent of all additional dimensions of the data resource using the uniform additional dimension schema (using `interval`, `crs/trs/vrs` and optionally `grid`).
Test method:	**Given:** a map resource that conformed successfully to /conf/core for which an extent description is available **When:** retrieving the description of the data resource **Then:** (assert that all parts of the requirement are fully satisfied)

A.9.2. Requirement general subsetting `subset` parameter

Test id:	/conf/general-subsetting/subset-definition
Requirement:	/req/general-subsetting/subset-definition
Test purpose:	Verify that the implementation supports general subsetting using the `subset` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps using the `subset` parameter for an additional dimension besides space and time **Then:** (assert that all parts of the requirement are fully satisfied)

A.10. Conformance Class "Coordinate Reference System"

A.10.1. Requirement `crs` parameter definition

Test id:	/conf/crs/crs-definition
Requirement:	/req/crs/crs-definition
Test purpose:	Verify that the implementation supports the output `crs` parameter for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps with the `crs` parameter for different available CRS and without **Then:** (assert that all parts of the requirement are fully satisfied)

A.10.2. Requirement CRS map success

Test id:	/conf/crs/map-success
Requirement:	/req/crs/map-success
Test purpose:	Verify that the implementation responds correctly to map requests using the `crs` parameter
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps with the `crs` parameter for different available CRS and without **Then:** (assert that all parts of the requirement are fully satisfied)

A.11. Conformance Class "Orientation"

A.11.1. Requirement `orientation` parameter

Test id:	/conf/orientation/orientation
Requirement:	/req/orientation/orientation
Test purpose:	Verify that the implementation supports the `orientation` parameter correctly for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps with the `orientation` parameter for different values and without **Then:** (assert that all parts of the requirement are fully satisfied)

A.11.2. Requirement orientation response headers

Test id:	/conf/orientation/response-headers

Requirement:	/req/orientation/response-headers
Test purpose:	Verify that the implementation includes the correct response headers for map requests using the `orientation` parameter.
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving maps with the `orientation` parameter for different values and without **Then:** (assert that all parts of the requirement are fully satisfied)

A.12. Conformance Class "Custom Projection CRS"

A.12.1. Requirement `crs-proj-method` parameter

Test id:	/conf/projection/crs-proj-method
Requirement:	/req/projection/crs-proj-method
Test purpose:	Verify that the implementation supports the `crs-proj-method` parameter correctly for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core and passing /conf/projections/projections-response **When:** retrieving maps with the `crs-proj-method` parameter for different available values as listed in `/projectionsAndDatums` **Then:** (assert that all parts of the requirement are fully satisfied)

A.12.2. Requirement `crs-proj-params` parameter

Test id:	/conf/projection/crs-proj-params
Requirement:	/req/projection/crs-proj-params
Test purpose:	Verify that the implementation supports the `crs-proj-params` parameter correctly for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core and passing /conf/projections/projections-response **When:** retrieving maps with the `crs-proj-method` parameter for different available values and different values of the associated method parameters (specified using the `crs-proj-params` query parameter) as listed in `/projectionsAndDatums` **Then:** (assert that all parts of the requirement are fully satisfied)

A.12.3. Requirement `crs-proj-center` parameter

Test id:	/conf/projection/crs-proj-center-definition
Requirement:	/req/projection/crs-proj-center-definition
Test purpose:	Verify that the implementation supports the `crs-proj-center` parameter correctly for map requests

Test method:	**Given:** a map resource that conformed successfully to /conf/core and passing /conf/projections/projections-response **When:** retrieving maps with the `crs-proj-method` parameter for different available values as listed in `/projectionsAndDatums` and the `crs-proj-center` parameter for different values **Then:** (assert that all parts of the requirement are fully satisfied)

A.12.4. Requirement `crs-datum` parameter

Test id:	/conf/projection/crs-datum
Requirement:	/req/projection/crs-datum
Test purpose:	Verify that the implementation supports the `crs-datum` parameter correctly for map requests
Test method:	**Given:** a map resource that conformed successfully to /conf/core and passing /conf/projections/projections-response **When:** retrieving maps with the `crs-datum` parameter for different available values as listed in `/projectionsAndDatums` **Then:** (assert that all parts of the requirement are fully satisfied)

A.12.5. Requirement custom CRS projection response headers

Test id:	/conf/projection/response-headers
Requirement:	/req/projection/response-headers
Test purpose:	Verify that the implementation responds to map requests using the `crs-proj-method` parameter and/or `crs-datum` with the correct response headers
Test method:	**Given:** a map resource that conformed successfully to /conf/core and passing /conf/projections/projections-response **When:** retrieving maps with the `crs-proj-method` parameter for different available values, different values of the associated method parameters (using both `crs-proj-center` and `crs-proj-params`), and different values for `crs-datum` as listed in `/projectionsAndDatums` **Then:** (assert that all parts of the requirement are fully satisfied)

A.12.6. Requirement `/projectionsAndDatums` resource

Test id:	/conf/projection/projections-resource
Requirement:	/req/projection/projections-resource
Test purpose:	Verify that the implementation supports retrieving the list of available projection operation methods, their parameters, and the list of available datums at `/projectionsAndDatums`
Test method:	**Given:** an API implementation being tested **When:** retrieving the `/projectionsAndDatums resource **Then:** (assert that all parts of the requirement are fully satisfied)

A.12.7. Requirement `/projectionsAndDatums` response

Test id:	/conf/projection/projections-response
Requirement:	/req/projection/projections-response
Test purpose:	Verify that the implementation responds correctly to a request for the `/projectionsAndDatums` resource, conforming to the JSON schema and using the correct URIs
Test method:	**Given:** an API implementation being tested passing /conf/projection/projections-resource **When:** retrieving the `/projectionsAndDatums resource` **Then:** (assert that all parts of the requirement are fully satisfied)

A.13. Conformance Class "Collection Map"

A.13.1. Requirement collection description links

Test id:	/conf/collection-map/desc-links
Requirement:	/req/collection-map/desc-links
Test purpose:	Verify that the implementation links correctly from the collection description resource to the map resource
Test method:	**Given:** a collection from an API implementation conforming to OGC API - Common - Part 2: Geospatial Data "Collections" conformance class **When:** retrieving the JSON representation of the description for that collection **Then:** (assert that all parts of the requirement are fully satisfied)

A.13.2. Requirement collection description CRS

Test id:	/conf/collection-map/desc-crs
Requirement:	/req/collection-map/desc-crs
Test purpose:	Verify that the implementation describes the supported CRS correctly in its collection description resources
Test method:	**Given:** an API implementation conforming to OGC API - Common - Part 2: Geospatial Data "Collections" conformance class **When:** retrieving the JSON representation of the description for that collection **Then:** (assert that all parts of the requirement are fully satisfied)

A.13.3. Requirement collection map operation

Test id:	/conf/collection-map/map-operation
Requirement:	/req/collection-map/map-operation
Test purpose:	Verify that the implementation supports retrieving maps from an OGC API a collection as defined in the OGC API – Common Standard.

Test method:	**Given:** a collection correctly linking to a map resource as per /conf/collection-map/desc-links **When:** retrieving a map for that collection resource as per /conf/core **Then:** (assert that all parts of the requirement are fully satisfied)

A.14. Conformance Class "Dataset map"

A.14.1. Requirement dataset landing page

Test id:	/conf/dataset-map/landingpage
Requirement:	/req/dataset-map/landingpage
Test purpose:	Verify that the implementation supports linking properly from an OGC API landing page to a map resource
Test method:	**Given:** a dataset provided by an API implementation conforming to OGC API - Common - Part 1: Core **When:** retrieving the JSON representation of the landing page description for that dataset **Then:** (assert that all parts of the requirement are fully satisfied)

A.14.2. Requirement dataset description extent

Test id:	/conf/dataset-map/desc-extent
Requirement:	/req/dataset-map/desc-extent
Test purpose:	Verify that the implementation describes the extent of the dataset correctly from the landing page
Test method:	**Given:** a dataset provided by an API conforming to OGC API - Common - Part 1: Core **When:** retrieving the JSON representation of the landing page description for that dataset **Then:** (assert that all parts of the requirement are fully satisfied)

A.14.3. Requirement dataset description CRS

Test id:	/conf/dataset-map/desc-crs
Requirement:	/req/dataset-map/desc-crs
Test purpose:	Verify that the implementation describes the supported CRS correctly in its landing page resource
Test method:	**Given:** a dataset provided by an API conforming to OGC API - Common - Part 1: Core **When:** retrieving the JSON representation of the landing page description for that dataset **Then:** (assert that all parts of the requirement are fully satisfied)

A.14.4. Requirement dataset map operation

Test id:	/conf/dataset-map/operation
Requirement:	/req/dataset-map/operation
Test purpose:	Verify that the implementation supports retrieving dataset maps a resource exposed by the OGC Maps API implementation
Test method:	**Given:** an OGC API dataset correctly linking to a map resource as per /conf/dataset-map/landingpage **When:** retrieving a map for that dataset resource as per /conf/core **Then:** (assert that all parts of the requirement are fully satisfied)

A.15. Conformance Class "Styled Map"

A.15.1. Requirement styled map links

Test id:	/conf/styled-map/desc-links
Requirement:	/req/styled-map/desc-links
Test purpose:	Verify that the implementation links correctly from a style resource to a map resource
Test method:	**Given:** a list of styles provided by an API implementation conforming to OGC API - Styles - Part 1: Core **When:** retrieving the JSON representation of that list of styles **Then:** (assert that all parts of the requirement are fully satisfied)

A.15.2. Requirement styled map operation

Test id:	/conf/styled-map/map-operation
Requirement:	/req/styled-map/map-operation
Test purpose:	Verify that the implementation supports retrieving maps from *OGC API - Styles* style resources
Test method:	**Given:** a style correctly linking to a map resource as per /conf/styled-map/desc-links **When:** retrieving a map for that style as per /conf/core **Then:** (assert that all parts of the requirement are fully satisfied)

A.16. Conformance Class "PNG"

A.16.1. Requirement PNG map content

Test id:	/conf/png/content
Requirement:	/req/png/content

Test purpose:	Verify that the implementation supports retrieving maps negotiating for PNG content
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving a PNG (`image/png`) representation of a map resource through HTTP content negotiation **Then:** (assert that all parts of the requirement are fully satisfied)

A.17. Conformance Class "JPEG"

A.17.1. Requirement JPEG map content

Test id:	/conf/jpeg/content
Requirement:	/req/jpeg/content
Test purpose:	Verify that the implementation supports retrieving maps negotiating for JPEG content
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving a JPEG (`image/jpeg`) representation of a map resource through HTTP content negotiation **Then:** (assert that all parts of the requirement are fully satisfied)

A.18. Conformance Class "JPEG XL"

A.18.1. Requirement JPEG XL map content

Test id:	/conf/jpegxl/content
Requirement:	/req/jpegxl/content
Test purpose:	Verify that the implementation supports retrieving maps negotiating for JPEG XL content
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving a JPEG XL (`image/jxl`) representation of a map resource through HTTP content negotiation **Then:** (assert that all parts of the requirement are fully satisfied)

A.19. Conformance Class "TIFF"

A.19.1. Requirement TIFF map content

Test id:	/conf/tiff/content
Requirement:	/req/tiff/content
Test purpose:	Verify that the implementation supports retrieving maps negotiating for TIFF and/or GeoTIFF content

Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving a TIFF (`image/tiff`) and GeoTIFF (`image/tiff; application=geotiff`) representation of a map resource through HTTP content negotiation **Then:** (assert that all parts of the requirement are fully satisfied)

A.20. Conformance Class "SVG"

A.20.1. Requirement SVG map content

Test id:	/conf/svg/content
Requirement:	/req/svg/content
Test purpose:	Verify that the implementation supports retrieving maps negotiating for SVG content
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving an SVG (`image/svg+xml`) representation of a map resource through HTTP content negotiation **Then:** (assert that all parts of the requirement are fully satisfied)

A.21. Conformance Class "HTML"

A.21.1. Requirement HTML map content

Test id:	/conf/html/content
Requirement:	/req/html/content
Test purpose:	Verify that the implementation supports retrieving maps negotiating for HTML content
Test method:	**Given:** a map resource that conformed successfully to /conf/core **When:** retrieving an (`text/html`) HTML representation of a map resource HTTP content negotiation **Then:** (assert that all parts of the requirement are fully satisfied)

A.22. Conformance Class "API Operations"

A.22.1. Requirement API Operations completeness

Test id:	/conf/api-operations/completeness
Requirement:	/req/api-operations/completeness
Test purpose:	Verify that the implementation completely and correctly describes the map resources

Test method:	**Given:** an API conforming to *OGC API - Common - Part 1: Core* "Landing Page" conformance class **When:** retrieving the API description **Then:** (assert that all parts of the requirement are fully satisfied)

A.22.2. Requirement API Operation identifiers

Test id:	/conf/api-operations/operation-id
Requirement:	/req/api-operations/operation-id
Test purpose:	Verify that the implementation uses the correct API operation identifier suffixes to identify the resources defined in the Maps API Standard
Test method:	**Given:** an API implementation conforming to *OGC API - Common - Part 1: Core* "Landing Page" conformance class supporting an API definition language with a concept of operation identifiers **When:** retrieving the API description **Then:** (assert that all parts of the requirement are fully satisfied)

A.23. Conformance Class "CORS"

A.23.1. Requirement CORS

Test id:	/conf/cors/cors
Requirement:	/req/cors/cors
Test purpose:	Verify that the implementation completely and correctly implement CORS
Test method:	**Given:** an API conforming to *OGC API - Common - Part 1: Core* "Landing Page" conformance class **When:** retrieving the API description **Then:** (assert that all parts of the requirement are fully satisfied)

Annex B: Examples (informative)

This annex provides a set of examples illustrating requests and responses for retrieving maps using capabilities defined by this Maps API.

B.1. Default map

The first example shows the response of a map endpoint with a request with no parameters (https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map).

The server is free to respond with any `bbox` and any `width` and `height`. In this case the server default behavior is to render the whole of Great Britain in a 631x1024 pixels canvas.

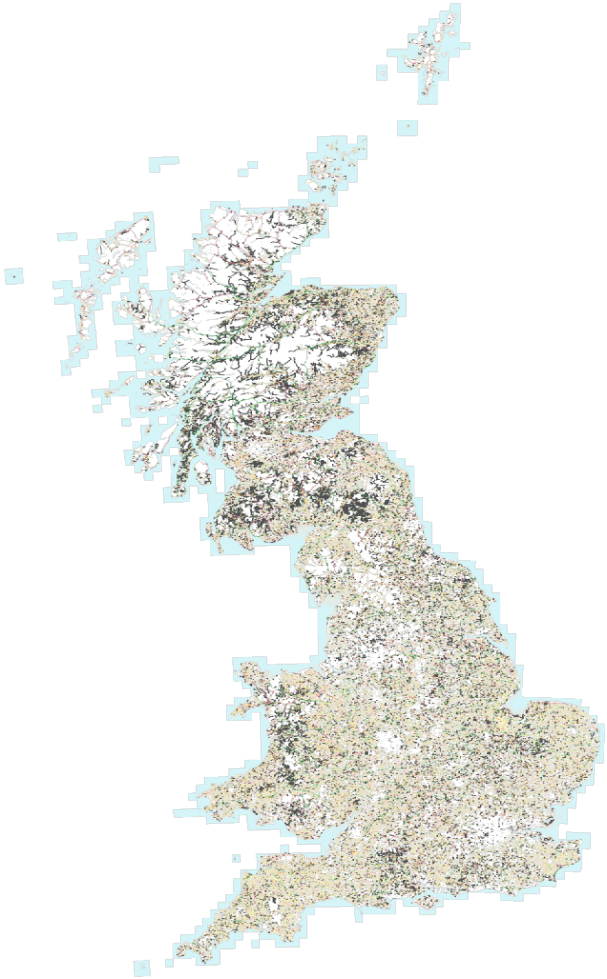


Figure 4. Great Britain data served as a map without specifying any parameter. From: OS OpenMap - Local [https://www.ordnancesurvey.co.uk/products/os-open-map-local], by the Ordnance Survey.

The headers of the response provide additional information on the bounding box (`Content-Bbox`) and the CRS of the image (`Content-Crs` is omitted here, indicating the default CRS84).

```
HTTP/1.1 200 OK
Expires: Tue, 19 Nov 2024 09:58:40 GMT
Access-Control-Allow-Origin: *
Vary: Accept, Accept-Encoding, Prefer
Content-Type: image/png
Content-Bbox: -8.756498,49.814737,1.848147,60.948016
Content-Length: 538037
```

B.2. Collection description

This map resource is attached to an *OGC API - Common - Part 2* [https://docs.ogc.org/DRAFTS/20-024.html] collection origin, which is available from the parent resource path:

https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal [https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal?f=json]

Example 11. JSON collection description response for the OS OpenMap Local collection


```
{
   "id" : "OpenMapLocal",
   "title" : "OS OpenMap - Local",
   "description" : "OS OpenMap - Local, by the OrdnanceSurvey",
   "attribution" : "<a href='https://www.ordnancesurvey.co.uk/products/os-open-
map-local'>OS OpenMap - Local</a>",
   "extent" : {
      "spatial" : {
         "bbox" : [ [ -8.7564981947375, 49.8147371614082,
                      1.8481470870055, 60.9480162305518 ] ]
   } },
   "minScaleDenominator" : 4265.4591676995678,
   "minCellSize" : 0.0000107288361,
   "storageCrs" : "https://www.opengis.net/def/crs/OGC/1.3/CRS84",
   "crs" : [
      "https://www.opengis.net/def/crs/OGC/1.3/CRS84",
      "https://www.opengis.net/def/crs/EPSG/0/4326",
      "https://www.opengis.net/def/crs/EPSG/0/3857",
      "https://www.opengis.net/def/crs/EPSG/0/3395"
   ],
   "links" : [
      { "rel" : "self", "title" : "Information about the OpenMapLocal data",
        "href" : "/ogcapi/collections/OpenMapLocal"
      },
      { "rel" : "[ogc-rel:map]", "title" : "Default map",
        "href" : "/ogcapi/collections/OpenMapLocal/map"
      },
      { "rel" : "[ogc-rel:tilesets-map]",
        "title" : "Map tilesets available for this collection",
        "href" : "/ogcapi/collections/OpenMapLocal/map/tiles"
      },
      { "rel" : "[ogc-rel:tilesets-vector]",
        "title" : "Multi-layer vector tilesets available for this collection",
        "href" : "/ogcapi/collections/OpenMapLocal/tiles"
      },
      { "rel" : "[ogc-rel:styles]", "title" : "Styles for OpenMapLocal",
        "href" : "/ogcapi/collections/OpenMapLocal/styles"
      }
   ]
}
```

B.3. Scaling and subsetting the map

The following examples are going to be centered on The O2 (formerly known as the Millenium Dome) shown here in a night aerial picture. Note that the roundness of the feature facilitates validating its aspect ratio in physical units.

Figure 5. The O2 dome peninsula (image captured from a plane by an editor of this standard while working on this annex)

The following map request only specifies a `center` point parameter next to the O2 Dome:

https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5

The server interprets the coordinates as CRS84 and decides to respond with a low scale denominator (high level of detail) suitable for the dataset and with reasonable default `width` and `height` (1024x1024 pixels). The response is shown in the following image.

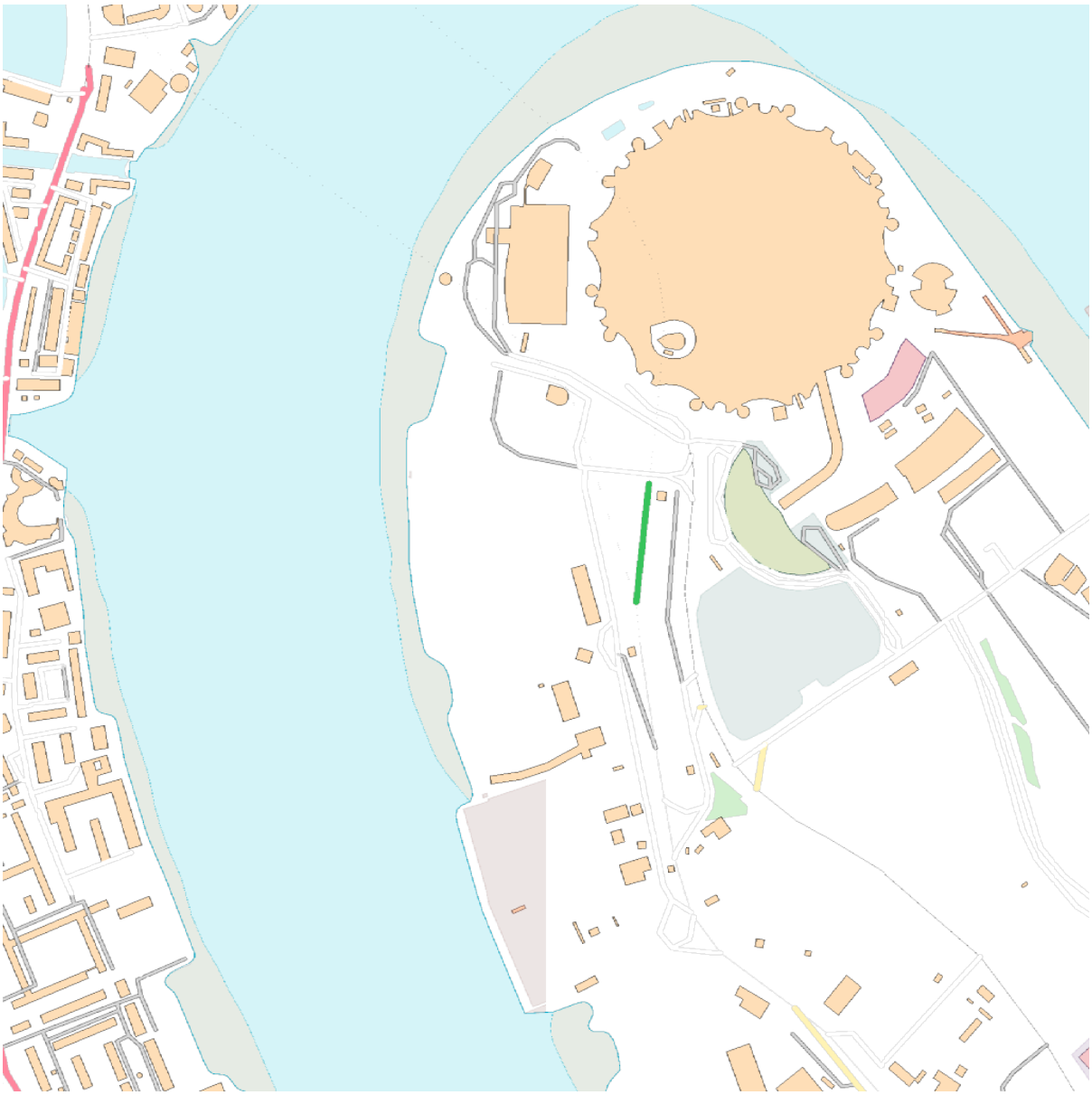


Figure 6. Map of OS OpenMap - Local close to The O2 dome, specifying `center` at 51.5°N, 0°E

The headers of the response provide additional information on the bounding box (`Content-Bbox`). Since the `Content-Crs` is not specified in this case, the client can assume CRS84.

```
HTTP/1.1 200 OK
Expires: Tue, 19 Nov 2024 09:57:36 GMT
Access-Control-Allow-Origin: *
Vary: Accept, Accept-Encoding, Prefer
Content-Type: image/png
Content-Bbox: -0.008805,51.494504,0.008805,51.505496
Content-Length: 188490
```

The following request is equivalent, using the value of that `Content-Bbox` as the value for the `bbox` parameter instead of using `center`, explicitly specifying the same `width` and `height` dimensions as those default values chosen by the server for the above request:

https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?bbox=-0.008805,51.494504,0.008805,51.505496&width=1024&height=1024

There is also an equivalent notation for the previous request that uses `subset` instead of `bbox`:

https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?subset=Lat(51.494504:51.505496),Lon(-0.008805:0.008805)&width=1024&height=1024

As explained in the scale and aspect ratio considerations section, clients wishing to retrieve identical responses from different implementations should specify either of these `bbox` or `subset` parameters, together with a `width` and `height`. A smaller image can be requested by specifying `height` of the image.

https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5&height=512

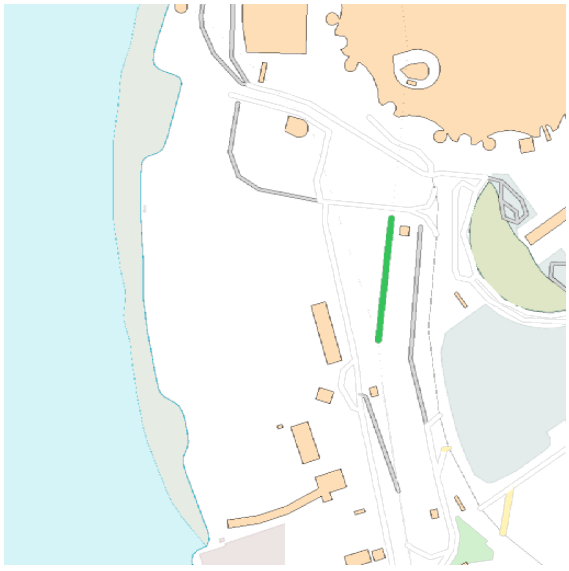


Figure 7. Map of OS OpenMap - Local centered on The O2 dome using `height=512`

The server would be free to act otherwise, but it automatically adjusted the width to also be 512. Notice that to preserve the same default scale with a smaller image, the spatial region (bounding box) was reduced accordingly. This behavior is particularly important when the client requests a specific scale, as in the following request which specifies the same `center` point parameter as before, but requests a map for a 1:8000 scale:

https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5&height=512&scale-denominator=8000

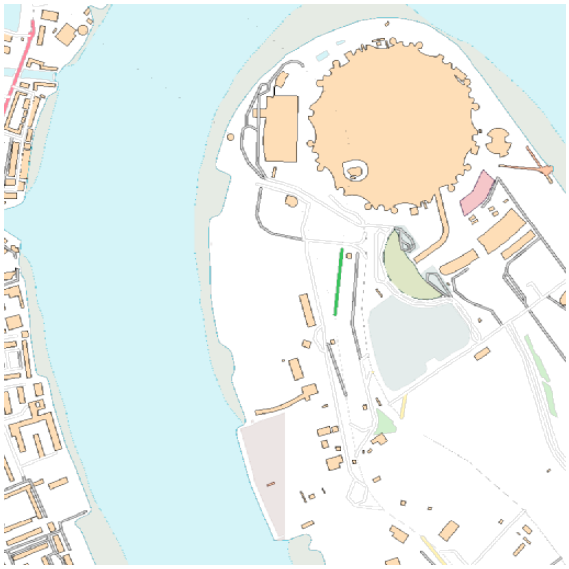


Figure 8. Map of OS OpenMap - Local centered on The O2 dome at 1:8000 scale using `scale-denominator`

The server responded with the same `width` and `height` (512x512 pixels). The headers of the response provide additional information on the bounding box of the image.

Clients can easily zoom in and out by simply changing the `scale-denominator` parameter as in the following images at different scales:

Figure 9. Map of OS OpenMap - Local centered on The O2 dome at 1:12,000 scale [https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5&height=512&scale-denominator=12000] using `scale-denominator`

Figure 10. Map of OS OpenMap - Local centered on The O2 dome at 1:20,000 scale [https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5&height=512&scale-denominator=20000] using `scale-denominator`

Figure 11. Map of OS OpenMap - Local centered on The O2 dome at 1:30,000 scale [https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5&height=512&scale-denominator=30000] using `scale-denominator`

Figure 12. Map of OS OpenMap - Local centered on The O2 dome at 1:50,000 scale [https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5&height=512&scale-denominator=50000] using `scale-denominator`

A rectangular image could be “forced” by also specifying the width of the image to be 1024, while keeping the rest of the parameters:

https://maps.gnosis.earth/ogcapi/collections/OpenMapLocal/map?center=0,51.5&scale-denominator=50000&width=1024&height=512

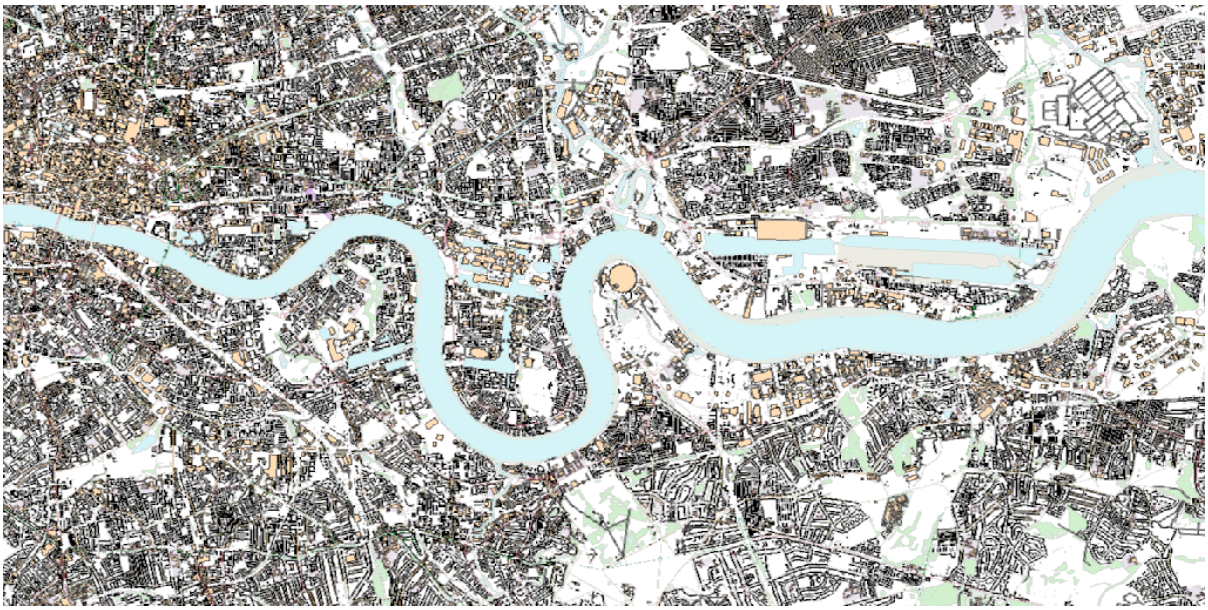


Figure 13. Wider 1024x512 map of OS OpenMap - Local centered on the O2 dome at 1:50,000 scale

For this last request, specifying both the `width` and `height`, the `center`, the `scale-denominator`, combined with the fact that the default value of `mm-per-pixel` is defined as 0.28 mm/pixel, defines all the parameters necessary to make the subsetting and scaling *mostly* predictable by the client. As explained in the scale and aspect ratio considerations section, implementations may compute the dimensions and bounding boxes not explicitly specified slightly differently. Because of these potential differences, clients should always consider the bounding box information in the response headers for georeferencing purposes, as well as the actual dimensions of the image returned. This will also avoid problems in cases where the server may decide to correct the center or bounding box due to the values being out of range. The `center` and `scale-denominator` parameters are primarily intended as convenience parameters to let the server automatically compute ideal bounding boxes and dimensions, while specifying a spatial region using the `bbox` or `subset` parameter as well as `width` and `height` should result in more deterministic responses.

B.4. Temporal subsetting

Spatial datasets are often also organized with a temporal dimension in addition to two or three spatial dimensions (some of these datasets are sometimes called time series or datacubes).

The following example reuses the same subsetting and scaling from the earlier rectangular 1:50,000 map of London, and applies it to a Sentinel-2 collection of images. The `datetime` parameter selects a particular day of the time series (April 1st, 2022).

https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/map?center=0,51.5&scale-denominator=50000&datetime=2022-04-01&width=1024&height=512

Figure 14. A map of Sentinel-2 data from April 1st, 2022 of the same area. From: Copernicus SENTINEL-2 operated by ESA [https://sentinel.esa.int/web/sentinel/missions/sentinel-2].

There is an equivalent notation for the previous request that uses `subset` for the *time* axis instead of the `datetime` parameter. This subsetting axis can also be combined within a single `subset` parameter value together with subsetting for the `Lat` and `Lon` axes, instead of using `center` and `scale-denominator`, or `bbox`. Note that in this case, the time string should be enclosed in double quotes.

https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/map?width=1024&height=512&subset=time("2022-04-01"),Lat(51.467787:51.532213),Lon(-0.103152:0.103152) [https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/map?width=1024&height=512&subset=time(%222022-04-01%22),Lat(51.467787:51.532213),Lon(-0.103152:0.103152)]

B.5. Styled maps

The following two example requests, for the same region and time of interest, illustrate two additional styles available from the same sentinel-2 datacube, in addition to its default Red, Green, Blue natural color style. The first style, symbolizes the Scene Classification Layer categories:

https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/styles/scl/map?center=0,51.5&scale-denominator=50000&datetime=2022-04-01&width=1024&height=512

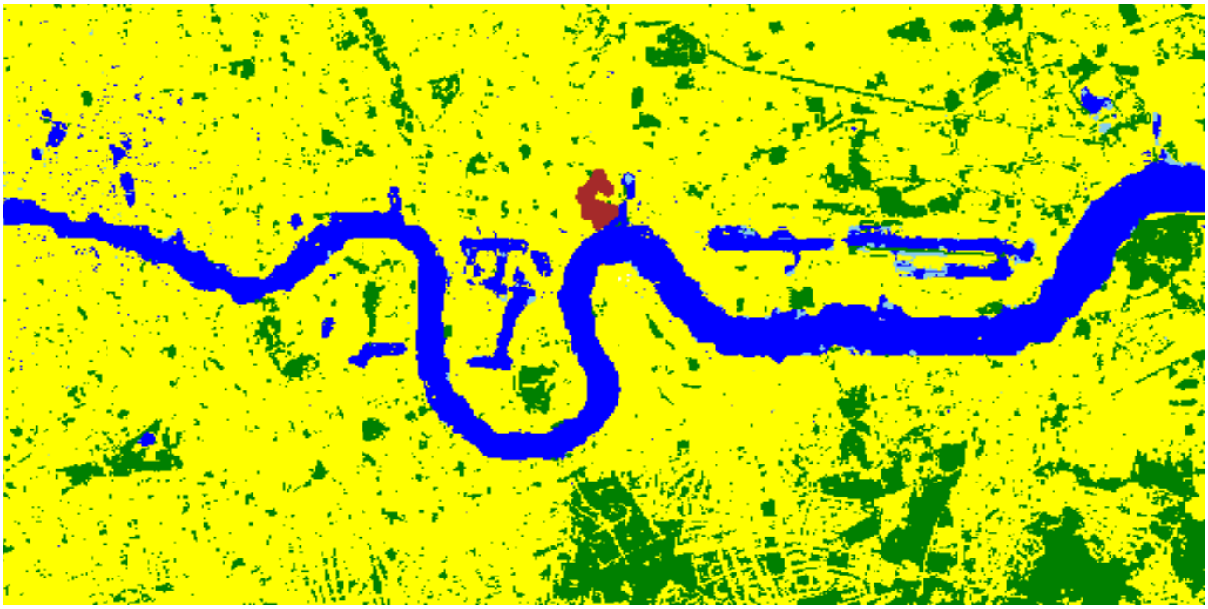


Figure 15. A map of a scene classification layer style for Sentinel-2 data from April 1st, 2022 of London. From: Copernicus SENTINEL-2 operated by ESA [https://sentinel.esa.int/web/sentinel/missions/sentinel-2].

The next style, using style identifier *evi2*, represents an Enhanced Vegetation Index (EVI) calculated from bands B02 (blue), B04 (red) and B08 (near infrared):

https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/styles/evi2/map?center=0,51.5&scale-denominator=50000&datetime=2022-04-01&width=1024&height=512

Figure 16. A map of an Enhanced Vegetation Index (EVI) style for Sentinel-2 data from April 1st, 2022 of London. From: Copernicus SENTINEL-2 operated by ESA [https://sentinel.esa.int/web/sentinel/missions/sentinel-2].

The following example requests illustrate how to retrieve two different styles for a High Resolution (1 m) Digital Terrain Model (DTM) of the Red River in Manitoba, from Natural Resources Canada. Styles with identifiers *style1* and *style2* are available at `⋯/styles/{styleId}`, through *OGC API - Styles* [https://docs.ogc.org/DRAFTS/20-009.html], and provide links to map resources.

https://maps.gnosis.earth/ogcapi/collections/HRDEM-RedRiver:DTM:1m/styles/style1/map?center=-97.06,49.937&scale-denominator=28000&height=600&width=1000

Figure 17. Styled map of High Resolution DTM [https://open.canada.ca/data/en/dataset/957782bf-847c-4644-a757-e383c0057995] from Natural Resources Canada (style1)

https://maps.gnosis.earth/ogcapi/collections/HRDEM-RedRiver:DTM:1m/styles/style2/map?center=-97.06,49.937&scale-denominator=28000&height=600&width=1000

Figure 18. Styled map of High Resolution DTM from Natural Resources Canada, showing alternative style2

B.6. Additional dimensions

It is also common for spatial datasets, especially for weather and climate data, to feature additional dimensions beyond space and time, such as pressure levels, or additional time dimensions relating to forecasting. These can all be handled in a generic manner also using the `subset` parameter. The following two example illustrates how to retrieve a map of the temperature for the whole world at *pressure* (an extra dimension) levels of 500 and 850 hectopascals:

https://maps.gnosis.earth/ogcapi/collections/climate:cmip5:byPressureLevel:temperature/map?subset=pressure(500)&datetime=2023-07-03&bgcolor=gray

Figure 19. A map of CMIP5 temperature data of the world at 500 hPa on July 3rd, 2023 (from Copernicus climate data store [https://cds.climate.copernicus.eu/cdsapp#!/dataset/projections-cmip5-daily-pressure-levels])

https://maps.gnosis.earth/ogcapi/collections/climate:cmip5:byPressureLevel:temperature/map?subset=pressure(850)&datetime=2023-07-03&bgcolor=gray

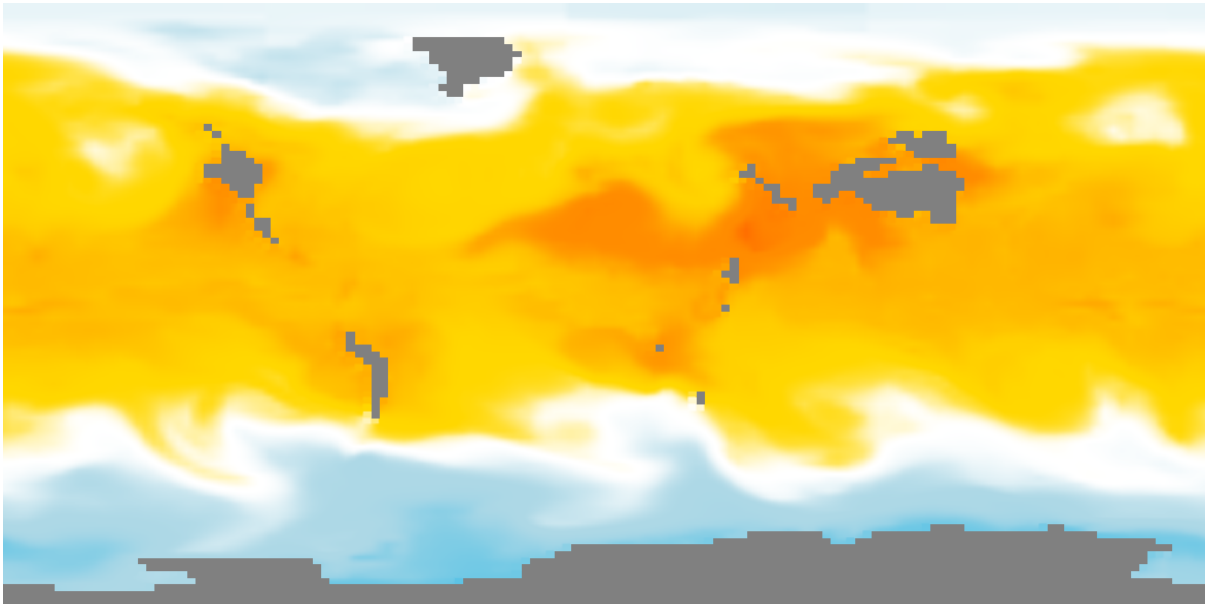


Figure 20. A map of CMIP5 temperature data of the world at 850 hPa on July 3rd, 2023 (from Copernicus climate data store [https://cds.climate.copernicus.eu/cdsapp#!/dataset/projections-cmip5-daily-pressure-levels])

The following examples illustrate how to retrieve a map of the relative humidity for the whole world at *pressure* (an extra dimension) levels of 500 and 975 hectopascals:

https://maps.gnosis.earth/ogcapi/collections/climate:era5:relativeHumidity/map?subset=pressure(500)&datetime=2023-04-06T23:00:00Z&bgcolor=skyBlue

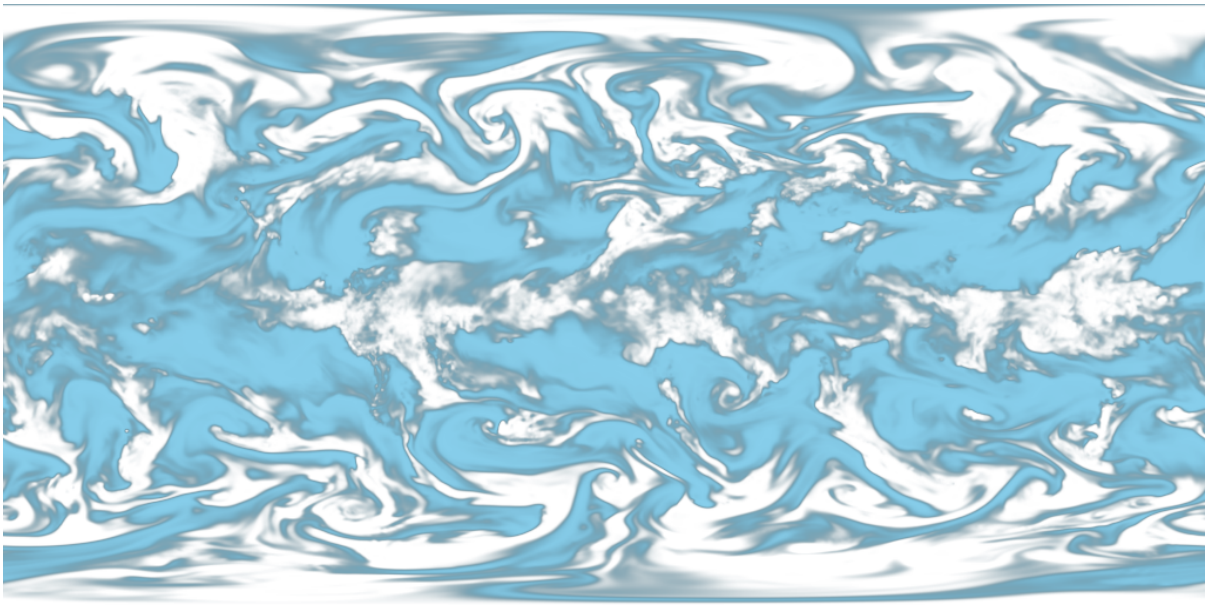


Figure 21. A map of ERA5 reanalysis data showing Relative Humidity of the whole world at 500 hPa on April 6th, 2023 at 23:00:00 UTC (from Copernicus climate data store [https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-pressure-levels])

https://maps.gnosis.earth/ogcapi/collections/climate:era5:relativeHumidity/map?subset=pressure(975)&datetime=2023-04-06T23:00:00Z&bgcolor=skyBlue

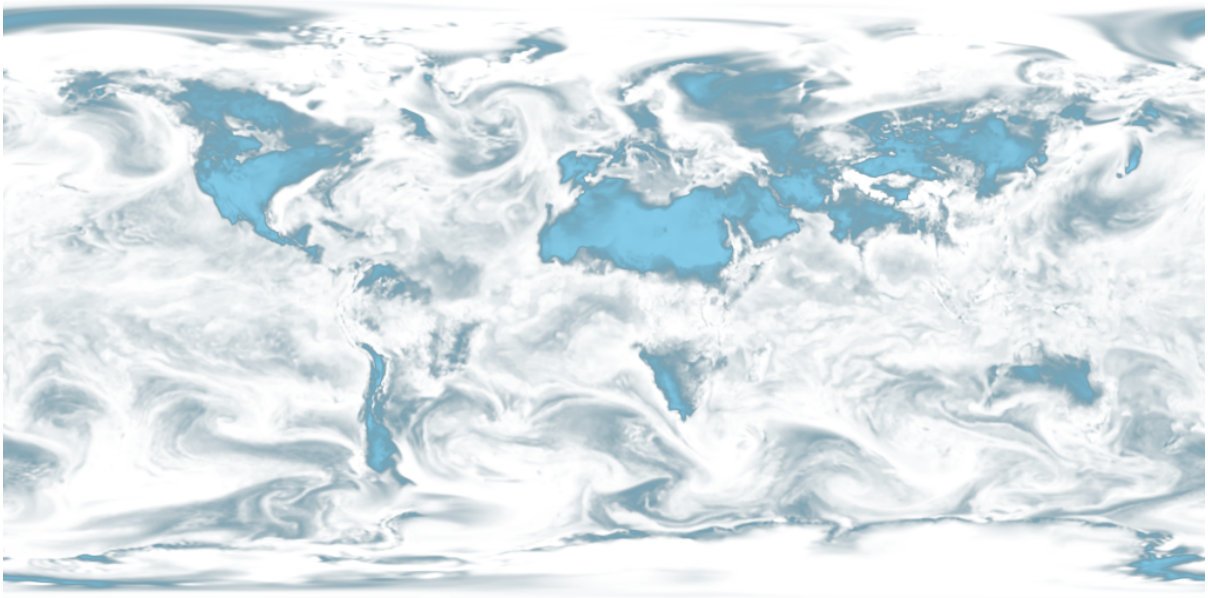


Figure 22. A map of ERA5 reanalysis data showing Relative Humidity of the whole world at 975 hPa on April 6th, 2023 at 23:00:00 UTC (from Copernicus climate data store [https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-pressure-levels])

The following JSON collection description [https://maps.gnosis.earth/ogcapi/collections/climate:era5:relativeHumidity?f=json] for these relative humidity examples illustrates how to describe the extent of multidimensional datasets, including the details of both regular and irregular grids.

Example 12. JSON collection description response for ERA5 relative humidity


```
{
   "id" : "climate:era5:relativeHumidity", "title" : "ERA5 Relative Humidity",
   "attribution" : "<a
href='https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-pressure-
levels'>Copernicus Climate Data Store</a>",
   "extent" : {
      "spatial" : {
         "bbox" : [ [ -180, -90, 180, 90 ] ],
         "grid" : [
            { "cellsCount" : 2049, "resolution" : 0.17578125 },
            { "cellsCount" : 1025, "resolution" : 0.17578125 }
         ]
      },
      "temporal" : {
         "interval" : [ [ "2023-04-01T00:00:00Z", "2023-04-06T23:00:00Z" ] ],
         "grid" : { "cellsCount" : 144, "resolution" : "PT1H" }
      },
      "pressure" : {
         "unit" : "hPa",
         "interval" : [ [ 1.0, 1000.0 ] ],
         "grid" : {
            "cellsCount" : 37,
            "coordinates" : [ 1.0, 2.0, 3.0, 5.0, 7.0, 10.0, 20.0,
               30.0, 50.0, 70.0, 100.0, 125.0, 150.0, 175.0, 200.0,
               225.0, 250.0, 300.0, 350.0, 400.0, 450.0, 500.0, 550.0,
               600.0, 650.0, 700.0, 750.0, 775.0, 800.0, 825.0, 850.0,
               875.0, 900.0, 925.0, 950.0, 975.0, 1000.0 ]
         }
      }
   },
   "minCellSize" : 0.17578125,
   "minScaleDenominator" : 69885283.0035897195339,
   "crs" : [
      "https://www.opengis.net/def/crs/OGC/1.3/CRS84",
      "https://www.opengis.net/def/crs/EPSG/0/4326",
      "https://www.opengis.net/def/crs/EPSG/0/3857",
      "https://www.opengis.net/def/crs/EPSG/0/3395"
   ],
   "storageCrs" : "https://www.opengis.net/def/crs/OGC/1.3/CRS84",
   "links" : [
      { "rel" : "self",
        "title" : "Information about the ERA5 Relative Humidity",
        "href" : "/ogcapi/collections/climate:era5:relativeHumidity"
      },
      { "rel" : "[ogc-rel:map]", "title" : "Default map",
        "href" : "/ogcapi/collections/climate:era5:relativeHumidity/map"
      },
      { "rel" : "[ogc-rel:tilesets-map]",
```

```
        "title" : "Map tilesets available for this collection",
        "href" : "/ogcapi/collections/climate:era5:relativeHumidity/map/tiles"
      },
      { "rel" : "[ogc-rel:styles]", "title" : "Styles for Relative Humidity",
        "href" : "/ogcapi/collections/climate:era5:relativeHumidity/styles"
      },
      { "rel" : "[ogc-rel:schema]", "title" : "Schema",
        "href" : "/ogcapi/collections/climate:era5:relativeHumidity/schema"
      },
      { "rel" : "[ogc-rel:coverage]", "title" : "Coverage for Relative Humidity",
        "href" : "/ogcapi/collections/climate:era5:relativeHumidity/coverage"
      },
      { "rel" : "[ogc-rel:tilesets-coverage]",
        "title" : "Coverage tilesets available for this collection",
        "href" : "/ogcapi/collections/climate:era5:relativeHumidity/tiles"
      }
   ]
}
```

B.7. Coordinate Reference Systems

While introducing the selection of an alternative output Coordinate Reference System (World Mercator, EPSG:3395) to the default native CRS (`storageCRS`) returned so far (CRS84), the following examples will zoom out significantly to a 1:20,000,000 scale. At the scales used in previous examples, the difference between those two CRSs would not be distinguishable, since the server automatically preserve scales in both dimensions, which makes the responses for those two CRSs almost visually equivalent on a local scale. These examples will illustrate the two CRS by requesting a map for the Blue Marble Next Generation (2004) from NASA Earth Observatory's Visible Earth, first explicitly requesting EPSG:4326 (whose main difference from CRS84 is that axis order is Latitude, Longitude).

https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?center=0,51.5&scale-denominator=20000000&crs=[EPSG:4326] [https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?center=0,51.5&scale-denominator=20000000&crs=%5BEPSG:4326%5D]

Notice that the response header now includes the `Content-Crs:` header, and that the `Content-Bbox:` axis order now follows the latitude, longitude order:

```
Content-Crs: <https://www.opengis.net/def/crs/EPSG/0/4326>
Content-Bbox: 25.729221,-28.573112,77.270779,28.573112
```

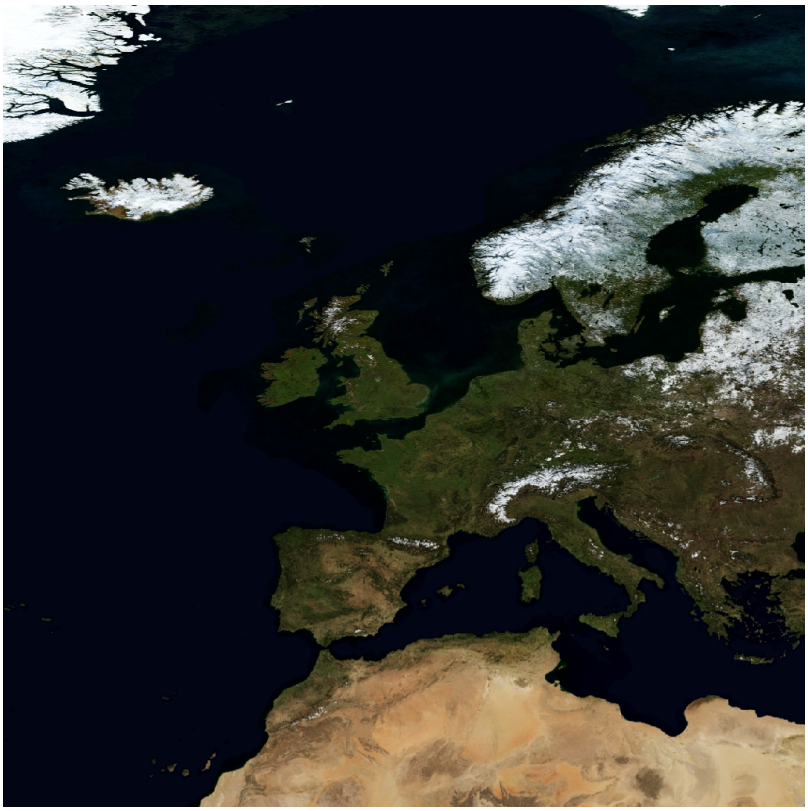


Figure 23. Map of NASA Earth Observatory's Blue Marble Next Generation (2004) [https://earthobservatory.nasa.gov/features/BlueMarble], in Plate Carrée (EPSG:4326) output `crs`

Now EPSG:3395 can be requested instead using:

https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?center=0,51.5&scale-denominator=20000000&crs=[EPSG:3395] [https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?center=0,51.5&scale-denominator=20000000&crs=%5BEPSG:3395%5D]

Figure 24. Map of NASA Earth Observatory's Blue Marble Next Generation (2004), using World Mercator (EPSG:3395) output `crs`

The `Content-Crs:` contains the coordinates of the bounding box selected from the requested scale and default dimensions, which can be used to make a request that will generate an equivalent response.

```
Content-Crs: <https://www.opengis.net/def/crs/EPSG/0/3395>
Content-Bbox: -4596385.263861,2080129.089271,4596385.263861,11273386.415933
```

In order to also specify the bounding box in that EPSG:3395 CRS, the following request also makes use of the `bbox-crs` parameter, which otherwise always defaults to CRS84 (regardless of the native CRS or selected output CRS).

https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?bbox-crs=[EPSG:3395]&bbox=-4596385.263861,2080129.089271,4596385.263861,11273386.415933&crs=[EPSG:3395]
[https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?bbox-crs=%5BEPSG:3395%5D&bbox=-4596385.263861,2080129.089271,4596385.263861,11273386.415933&crs=%5BEPSG:3395%5D]

B.8. Calculations to infer appropriate dimensions

In these examples, the client specifies a bounding box (using `bbox` or `subset`) from 30°N to 50°N and 0°E to 30°E and a `scale-denominator` of 1:10,000,000. The server needs to compute appropriate map image dimensions from these parameters that are provided (no default dimensions or center are used). The default 0.28 mm/pixel display resolution is used since `mm-per-pixel` is not specified. Two examples are given, one in a geographic Plate Carrée CRS (EPSG:4326) and one in a projected World Mercator CRS (EPSG:3395) (which uses a bounding box of the same geographic area transformed into coordinates in that CRS).

Regardless of the CRS, the number of physical meters that each pixel should represent can be computed with:

```
physicalMetersPerPixel = (mm-per-pixel / 1000 mm/m) * scale-denominator
physicalMetersPerPixel = (0.28 mm/pix / 1000 mm/m) * 10,000,000 = 2800 m/pix
```

B.8.1. Plate Carrée (EPSG:4326) Example

The first example is for an EPSG:4326 Plate Carrée CRS.

```
GET /collections/blueMarble/map?
   subset=Lat(30:50),Lon(0:30)&
   scale-denominator=10000000&
   crs=[EPSG:4326]

GET /collections/blueMarble/map?
   bbox=0,30,30,50&
   scale-denominator=10000000&
   crs=[EPSG:4326]
```

The latitude delta is 20°, whereas the longitude delta is 30°.

The implementation could assume the WGS84 111,319.49 meters / degree of latitude (`metersPerDegLat` below), and use the most equatorial latitude of the subset (30°N) to compute the numbers of meters / degree of longitude with:

```
metersPerDegLon = metersPerDegLat * cos(mostEquatorialLat)
metersPerDegLon = 111,319.49 m/deg * cos(30°) = 96,405.51 m/deg
```

The dimensions can then simply be computed (rounding to the nearest integer) with:

```
width  = deltaLon * metersPerDegLon / physicalMetersPerPixel
height = deltaLat * metersPerDegLat / physicalMetersPerPixel

width  = 30 deg *  96,405.51 m/deg / 2800 m/pix = 1033 pixels
height = 20 deg * 111,319.49 m/deg / 2800 m/pix =  795 pixels
```

B.8.2. World Mercator (EPSG:3395) Example

This second example is for the same geographical area, but for an EPSG:3395 World Mercator CRS instead.

```
GET /collections/blueMarble/map?
   subset=E(0:3339584.72),N(3482189.09:6413524.59)&
   scale-denominator=10000000&
   subset-crs=[EPSG:3395]&
   crs=[EPSG:3395]

GET /collections/blueMarble/map?
   bbox=0,3482189.09,3339584.72,6413524.59&
   scale-denominator=10000000&
   crs=[EPSG:3395]&
   bbox-crs=[EPSG:3395]
```

The easting delta is 3,339,584.72, whereas the northing delta is 2,931,335.50.

To correctly apply the scale, the ratio between CRS units and physical meters must be considered. This could be calculated for the center point: (E: 1,669,792.36, N: 4,947,856.84) which corresponds to (40.7514917°N, 15°E). One approach could be to project two points 1 degree of longitude apart around the center point, using the implementation's projection library, to obtain the number of CRS units per degree of longitude. Transforming (40.7514917°N, 14.5°E) to EPSG:3395 yields (E: 1,614,132.62, N: 4,947,856.85) and transforming (40.7514917°N, 15.5°E) yields (E: 1,725,452.11, N: 4,947,856.85). The resulting easting delta (`oneDegEastingDelta` below) is 111,319.49, which in this case can be recognized as the actual physical meters per degree at the equator, rather than at the center latitude used. Therefore, in this case implementations effectively needs to apply the reverse correction that had to be applied for Plate Carrée when considering the numbers of true meters per Easting unit. First, `metersPerDegLon` can be computed as in the previous example:

```
metersPerDegLon = metersPerDegLat * cos(centerLat)
metersPerDegLon = 111,319.49 m/deg * cos(40.7514917°) = 84,329.856 m/deg
```

Then the meters per easting unit can be computed:

```
metersPerEastingUnit = metersPerDegLon / oneDegEastingDelta
metersPerEastingUnit = 84,329.856 m/deg / 111,319.49 m/deg = 0.75754799
```

Notice that in this particular case of World Mercator, this is simply:

```
metersPerEastingUnit = cos(centerLat)
metersPerEastingUnit = cos(40.7514917°) = 0.75754799
```

The implementation could also apply similar logic to compute the meters per northing unit:

```
metersPerNorthingUnit = metersPerDegLat / oneDegNorthingDelta
```

However, in the case of World Mercator, it could simply assume that `metersPerNorthingUnit` is equal to `metersPerEastingUnit`.

Finally, the map image dimensions can be computed with:

```
width  = deltaEasting  * metersPerEastingUnit  / physicalMetersPerPixel
height = deltaNorthing * metersPerNorthingUnit / physicalMetersPerPixel

width  = 3,339,584.72 m * 0.75754799 / 2800 m/pix = 904 pixels
height = 2,931,335.50 m * 0.75754799 / 2800 m/pix = 793 pixels
```

B.9. Calculations to infer appropriate bounding boxes

In these examples, the client specifies a `center`, a `scale-denominator`, and optionally `width` and / or `height` dimensions. If either the `width` or `height` is not specified, the server could pick default values, such as making the missing dimension equal to the one provided or selecting default values. The examples will assume that the client explicitly requested a 1024 x 768 map at a 1:10,000,000 scale for a location centered on (41.8902°N, 12.4922°E).

To compute the extent in CRS units, first the physical meters per pixel can be computed using the same formula as earlier (and same result in this case):

```
physicalMetersPerPixel = (mm-per-pixel / 1000 mm/m) * scale-denominator
physicalMetersPerPixel = (0.28 mm/pix / 1000 mm/m) * 10,000,000 = 2800 m/pix
```

B.9.1. Plate Carrée (EPSG:4326) Example

This first example requests a map in an EPSG:4326 output CRS, using that same CRS for specifying the center as well:

```
GET /collections/blueMarble/map?
   center=41.8902,12.4922&
   center-crs=[EPSG:4326]&
   scale-denominator=10000000&
   crs=[EPSG:4326]&
   width=1024&
   height=768
```

A simple approach to computing the bounding box is to extend away from the center in both directions by the distance in CRS units corresponding to half the respective pixel dimension.

To consider the latitude of the subsets for computing the longitude extent, the latitude extent will be computed first, using the inverse of the earlier height computation:

```
deltaLat = height * physicalMetersPerPixel / metersPerDegLat
deltaLat = 768 pix * 2800 m/pix / 111,319.49 m/deg = 19.317372 degrees
```

A constant 111,319.49 meters / degree of latitude is assumed here again, rather than the more accurate polynomial equation taking into consideration the ellipsoid eccentricity mentioned earlier, which would yield slightly different results.

The lower and upper latitudes of the bounding can then be easily computed by adding and subtracting half this delta to the requested center latitude:

```
lowerLat = 41.8902°N - 19.317372 deg / 2 = 32.231514°N
upperLat = 41.8902°N + 19.317372 deg / 2 = 51.548886°N
```

From this, the most equatorial latitude can be established to be 32.231514°N, which can then be used to compute the meters per degrees of longitude, exactly like for the dimension computation examples:

```
metersPerDegLon = metersPerDegLat * cos(mostEquatorialLat)
metersPerDegLon = 111,319.49 m/deg * cos(32.231514°) = 94,165.15 m/deg
```

and the longitude delta can then be computed similarly:

```
deltaLon = width * physicalMetersPerPixel / metersPerDegLon
deltaLon = 1024 pix * 2800 m/pix / 94,165.15 m/deg = 30.448632 degrees
```

and from this the left (West) and right (East) longitude bounds:

```
leftLon  = 12.4922°E - 30.448632 deg / 2 =  2.732116°W
rightLon = 12.4922°E + 30.448632 deg / 2 = 27.716516°E
```

which completes the bounding box calculation. Expressed in the default CRS84 (longitude, latitude) order, the `bbox` parameter would be:

`bbox=-2.732116,32.231514,27.716516,51.548886`.

B.9.2. World Mercator (EPSG:3395) Example

This second example requests a map in an EPSG:3395 output CRS, using that same CRS for specifying the center as well:

```
GET /collections/blueMarble/map?
   center=1390625.34,5116008.23&
   center-crs=[EPSG:3395]&
   scale-denominator=10000000&
   crs=[EPSG:3395]&
   width=1024&
   height=768
```

The center corresponds to the same point as the previous example (41.8902°N, 12.4922°E).

Using the same approach as for the earlier dimension computation examples, transforming test points one degree apart, the number of physical meters per easting and northing units can be computed specifically for the requested center point (E: 1390625.34, N: 5116008.23). Like earlier, in the specific case of World Mercator the `oneDegEastingDelta` and `oneDegNorthingDelta` is constant at 111,319.49 m/deg, which corresponds to the number of meters per degree at the equator. The number of physical meters per degree of longitude at the center latitude (41.8902°N) can be computed in the same way as previous examples:

```
metersPerDegLon = metersPerDegLat * cos(centerLat)
metersPerDegLon = 111,319.49 m/deg * cos(41.8902°) = 82,869.096 m/deg
```

Then the meters per easting unit can be computed:

```
metersPerEastingUnit = metersPerDegLon / oneDegEastingDelta
metersPerEastingUnit = 82,869.096 m/deg / 111,319.49 m/deg = 0.74442576
```

Again, in the case of World Mercator, this is simply:

```
metersPerEastingUnit = cos(centerLat)
metersPerEastingUnit = cos(41.8902°) = 0.74442576
```

The implementation could also apply similar logic to compute the meters per northing unit:

```
metersPerNorthingUnit = metersPerDegLat / oneDegNorthingDelta
```

And again, in this case it could simply assume that `metersPerNorthingUnit` is equal to `metersPerEastingUnit`.

Then the delta easting and northing can be computed using the inverse of the equations used for computing dimensions:

```
deltaEasting  = width  * physicalMetersPerPixel / metersPerEastingUnit
deltaNorthing = height * physicalMetersPerPixel / metersPerNorthingUnit

deltaEasting  = 1024 pix * 2800 m/pix / 0.74442576 = 3,851,559.355 m
deltaNorthing =  768 pix * 2800 m/pix / 0.74442576 = 2,888,669.516 m
```

and finally this bounding box can be extended away from the requested center point (E: 1,390,625.34, N: 5,116,008.23):

```
leftEasting   = 1,390,625.34 - 3,851,559.355 / 2 =  -535,154.34
lowNorthing   = 5,116,008.23 - 2,888,669.516 / 2 = 3,671,673.47
rightEasting  = 1,390,625.34 + 3,851,559.355 / 2 = 3,316,405.02
upperNorthing = 5,116,008.23 + 2,888,669.516 / 2 = 6,560,342.99
```

which completes the bounding box calculation. Expressed in EPSG:3395 together with the required `bbox-crs`, the parameters would be:

`bbox=-535154.34,3671673.47,3316405.02,6560342.99&bbox-crs=[EPSG:3395]`.

Annex C: Evolution from OGC Web Map Service

C.1. From OGC Web Services to OGC Web APIs

OGC Web Service (OWS) Standards have historically implemented a Remote-Procedure-Call-over-HTTP architectural style using Extensible Markup Language (XML) for the metadata and capability payloads. This was state-of-the-art when initial versions of OGC Web Services were originally designed in the late 1990s and early 2000s. The resource-oriented RESTful Web API style is proposed as an alternative architectural style to the RPC pattern which was service-oriented. The Web Map Service 1.3 Standard did not describe a resource oriented architectural style or a Web API definition. This *OGC API - Maps* Standard specifies requirements for a Web API that follows the Web architecture and the W3C/OGC best practices for sharing Spatial Data on the Web [https://www.w3.org/TR/sdw-bp/] as well as the W3C best practices for sharing Data on the Web [https://www.w3.org/TR/dwbp/].

The *OGC API - Maps - Part 1: Core* Standard provides the necessary elements to incorporate ***maps*** support into a broader Web API implementation. These elements can be incorporated in an API based on *OGC API - Features - Part 1: Core* or can be incorporated in a Web API implementation based on *OGC API - Common - Part 1: Core*. Both specify a kernel of a Web API approach to services that follows current resource-oriented architecture practices in the OGC. The *OGC API - Common* Standard provides the foundation upon which implementations of the OGC API Standards can be built. *OGC API - Common* can be combined with the *Maps API* Standard and other resource-specific OGC API Standards to build an OGC standards-based API implementation. However, the *Maps API* Standard is specified in a way that can extend *OGC API - Common* but does not make *OGC API - Common* mandatory. In this way, the *Maps API* Standard can be reused as a building block in other APIs that do not follow the OGC API pattern.

Beside the general alignment with the architecture of the Web (e.g., consistency with HTTP/HTTPS, hypermedia controls), another goal for OGC API Standards is modularization. This goal has several facets:

- Clear separation between Core requirements and more advanced capabilities. The *OGC API - Maps - Part 1: Core* Standard presents the requirements that are relevant for almost any organization wanting to share or use maps at a fine-grained level. Additional capabilities deemed useful for several communities will be specified as extensions to the Core API.
- Technologies that change more frequently are decoupled and specified in separate modules ("requirements classes" in OGC terminology). This enables, for example, the use/re-use of new encodings for spatial data or Web API definition (such as new version of the OpenAPI description document).
- Modularization is not just about a single "service". OGC API Standards define building blocks that can be reused in APIs implementation in general. In other words, a server supporting *OGC API - Maps* should not be seen as a standalone service. Rather it should be viewed as a collection of Web API building blocks that together implement *OGC API - Maps* capabilities. A corollary for this is that it should be possible to implement an API that simultaneously conforms to conformance classes from the Features, Coverages, Maps, Tiles, and other future OGC API

standards.

This approach intends to support two types of client developers:

- Those that have never heard about the OGC. Developers should be able to create a client using the Web API definition without the need to adopt a specific OGC approach. For example, developers no longer need to read how to implement a *GetCapabilities* document, allowing them to focus on the geospatial aspects.
- Those that want to write a "generic" client that can access implementations of OGC API Standards. In other words, common capability and metadata resources are not specific to a particular API.

As a result of following a RESTful approach, OGC API implementations are not backwards compatible with OWS implementations per se. However, a design goal is to define OGC API Standards in a way that an OGC API interface can be mapped to an OWS implementation (where appropriate). OGC API Standards are intended to be simpler and more modern, but still an evolution from the previous versions and their implementations making the transition easy (e.g., by initially implementing facades in front of the current OWS services).

C.2. Comparison between *WMS* and *OGC API - Maps*

The OGC WMS and WMTS share the concept of a map and the capability to create and distribute maps at a limited resolution and size. In WMS, the number of rows and columns can be selected by the user within limits and in WMTS the number of rows and columns of the response is predefined in the tile matrix set.

The concept of a map used in OGC API Standards is more abstract than the one used in WMS. Read more about the map concept in the Overview.

C.2.1. Functionality changes from WMS

One important enhancement in the *OGC API - Maps* Standard when compared to WMS is that requests are built out of modular building blocks, allowing different levels of complexity: From a simple map request with no parameters to a precise selection of collections, spatiotemporal subsets, output dimensions, scale, background settings and so on.

Additionally, to accommodate different client preferences and use cases as well as maintain consistency with other OGC API data access mechanisms such as the *OGC API - Coverages* and *OGC API - Features* Standards, multiple equivalent subsetting mechanisms are supported such as subsetting by bounding box, date and time, generic dimension subset or center point.

Similarly, the ability to explicitly specify a scale denominator as a way to control down-sampling is also defined.

The concept of a display resolution (millimeters per pixel) is also introduced as an alternative to the default 0.28 mm/pixel specified in WMS, enabling a better control on how a server renders symbology suitable for devices and printed maps of different resolutions.

A flexible mechanism to define custom CRS and re-orient the map is also introduced, offering a

more comprehensive approach based on coordinate operation (see https://docs.ogc.org/as/18-005r4/18-005r4.html, definition 3.1.8) methods, parameters and datums (implying a particular ellipsoid) compared to the *AUTO2:* namespace for CRSs as specified in the WMS Standard. The *AUTO2* namespace is used for “automatic” coordinate reference systems.

Finally, *OGC API - Maps* can also be used to represent a vertical dimension. This can be done either by returning a 3D data format, by selecting a vertical subset of features to be rendered on a 2D output, or by rendering a vertical profile. Some examples include:

- 3D map to be displayed on virtual reality devices,
- 1D graph of NDVI over time for a particular point,
- 1D graph of a temperature over a vertical profile for a given spatial point,
- intensity of rain along a latitude and vertical dimension,
- video over temporal axis,
- video of temperature representing different elevation levels in different frames of the video.

C.2.2. Correspondence between WMS map metadata and OGC API Standards

In the WMS Standard, the `GetCapabilities` response provides some metadata about the server and individual `layer` sections that inform the client about layer characteristics and some restrictions useful to formulate a successful map query. In the OGC API - Maps Standard, the equivalent metadata is provided via the landing page, the list of collections, the collections details, the API definition, and the service-meta link from the landing page. Implementers of Web APIs are encouraged to make use of the mechanisms provided by other standards of the OGC API family to communicate the relevant metadata to the client.

The following table provides a reference to where some of metadata aspects at the service level are specified.

Table 13. Where some "service" metadata elements are specified

Name in <Service> WMS 1.3	**Where in the API**	**property**	**Specified in**
Title	service metadata[2]	title	OGC API - Common - Part 1
Name fixed to "WMS"	N/A		
Abstract	service metadata[2]	description	OGC API - Common - Part 1
OnlineResource	landing page	links	OGC API - Common - Part 1
Keywords	service metadata[2]	x-keywords	OGC API - Common - Part 1, Annex B.4
LayerLimit	service metadata[2]	x-OGC-limits.maps.maxCollections	This standard

Name in <Service> WMS 1.3	Where in the API	property	Specified in
MaxWidth MaxHeight	service metadata	x-OGC-limits.maps.maxWidth x-OGC-limits.maps.maxHeight x-OGC-limits.maps.maxPixels [1]	This standard Requirement Class "Scaling"
Fees	N/A		
AccessConstraints	N/A		

[1] `x-OGC-limits.maps.maxWidth`, `x-OGC-limits.maps.maxHeight` and `x-OGC-limits.maps.maxPixels` are intended to control the work load of the server by providing limitations in size of the outputs of the subset. `width` and `height` parameters in *OGC API - Maps* (defined in Requirement Class "Scaling") control the size of the response and its resolution. The core of OGC API - Maps does not provide explicit limits on the size and resolution, but the server is free to respond with an error to avoid work overload. `width` and `height` parameters are commonly related to the size of the device screen. The fact that new devices are being built with more and more available display pixels. As such, specifying a reasonable limit on the server side based on today's technology may become too restrictive for future devices. [2] service metadata may be provided as an extension of the `info` section of the Open API document as indicated in OGC API - Common - Part 1, Annex B.4

The following table provides a reference to where some of layer metadata aspects are specified.

Table 14. Where some "layer" metadata elements are specified

Name in WMS 1.3 <Layer>	Where in the API	property	Specified in
Title	collection response	title	OGC API - Common - Part 2
Name	collection response	id	OGC API - Common - Part 2
Abstract	collection response	description	OGC API - Common - Part 2
Keywords	collection response	keywords	OGC API - Records (Local Resources Catalogue)
Style	style response	id	OGC API - Styles - Part 1
EX_GeographicBoundingBox	collections response	extent.spatial	OGC API - Common - Part 2
CRS	collection response	storageCrs	This standard
BoundingBox	collection response	storageCrsExtent[1]	This standard

Name in WMS 1.3 <Layer>	Where in the API	property	Specified in
minScaleDenominator maxScaleDenominator	collection response	minScaleDenominator maxScaleDenominator	Possibly in OGC API - Common - Part 2
Sample Dimensions	collection response	extent [2]	This standard
MetadataURL	collection response	link with rel describedBy	OGC API Common - Part 2
Attribution	collection response	attribution	OGC API - Common - Part 2
Identifier AuthorityURL	collection response	externalIds	OGC API - Records (Local Resources Catalogue)
FeatureListURL	items response		OGC API - Features provides this capability
DataURL			OGC API - Features, Coverages and EDR provide download capabilities
queryable [3]			OGC API - Features, Coverages and EDR provide query capabilities
cascaded [4] noSubsets [5] fixedWidth [6] fixedHeight [7]	N/A		

Name in WMS 1.3 <Layer>	Where in the API	property	Specified in

[1] In WMS it was possible to specify one bounding box for each supported output CRS. In OGC API - Maps, it is only provided for the native CRS (storageCrs).

[2] If extra dimensions are supported the range of values are defined in additional properties of the 'extent' of the collection.

[3] No equivalent functionality to GetFeatureInfo is provided so this flag is not applicable to OGC API - Maps. Please use the query capabilities of other OGC API Standards instead.

[4] The `cascaded` XML attribute in WMS is removed because no practical use has been seen. `cascaded` indicated if a map was generated by the addressed service or by another service assisting the first one.

[5] The `noSubsets` XML attribute in WMS was used to indicate lack of subsetting support. The client will know if the server does not support the *spatial subsetting*, *date and time* (for temporal subsetting) or *general subsetting* conformance class by inspecting its conformance declaration.

[6] The `fixedWidth` XML attribute in WMS was used to indicate lack of scaling support. The client will know if the server does not support the *scaling* conformance class by inspecting its conformance declaration.

[7] The `fixedHeight` XML attribute in WMS was used to indicate lack of scaling support. The client will know if the server does not support the *scaling* conformance class by inspecting its conformance declaration.

NOTE The supported formats for map resources, or more precisely the media types of the supported encodings, can also be determined from the API definition. The desired encoding is selected using HTTP content negotiation. In addition to the parameters specified by the core, other parameters should be added.

NOTE The `opaque` XML attribute in WMS was rarely useful and has been removed. This attribute indicated whether the map data represents features that probably do not completely fill space and shows the background opaque (true) or transparent (false).

C.2.3. No equivalent to *GetFeatureInfo* as part of the OGC API - Maps - Part 1

The OGC Web Map Service *GetFeatureInfo* operation provides the capability for clients to implement some simple level of user interaction with the map. In essence the user can focus on a point in the map (e.g., by clicking on it) and the client will request from the server some textual information related to the elements represented at that point of the map (a functionality sometimes called "query by location"). If the elements represented in the map are simple features, the result should be related to their properties (attributes). If the map represents a coverage, the result should report the value of the coverage in that position (eventually, if the coverage is multidimensional, it

could be a e.g., time series graphic or a vertical profile). The format of the actual response is left to the discretion of the server.

GetFeatureInfo was first proposed in the 2000 version of the OGC Web Map Service Standard. In that environment *GetFeatureInfo* provided an easy to implement solution for the first step to "queryable" maps.

The new OGC API Standards emerged in a completely different context where most web content is dynamic and JavaScript is now a powerful programming language for the Web. Most simplistic implementations of WMS *GetFeatureInfo* resulted in an imperfect presentation of the attribute text. Users demand much more than query by location. Now, the integration of the different building blocks defined in OGC API Standards can be provided by default. A map is connected to a collection (or a dataset) that is probably also offered as features with *OGC API - Features* or as coverage with *OGC API - Coverages* — all from the same API landing page. Furthermore, *OGC API - Environmental Data Retrieval (EDR)* also provides a point query, similar to *GetFeatureInfo* as well as much more advanced queries by polygons, trajectories or corridors.

Implementers of map clients are encouraged to implement support for OGC API Standards in addition to *OGC API - Maps* to provide a functionality similar to *GetFeatureInfo*. Instead of building a request to a map point in map coordinates (I, J), implementers should use point narrow bounding boxes in CRS coordinates. For example:

- In *OGC API - Features*, map coordinates should be transformed to Lon,Lat WGS84 in the client side and implement a HTTP GET request to `/collections/{collectionId}/items?bbox=Lon,Lat,Lon,Lat`.
- In *OGC API - Coverages*, map coordinates should be transformed to native coordinates and use `/collections/{collectionId}/coverage?bbox=x,y,x,y` or the equivalent "subset" query.
- In *OGC API - EDR*, map coordinates should be transformed to a CRS coordinates and use `/collections/{collectionId}/position?coords=POINT(x y)` or by adding a radius query `/collections/{collectionId}/radius?coords=POINT(x y)&within=20&within-units=km`.

The use of *OGC API - Tiles* and serving vector tiled content directly also makes creating visualizations with query capabilities directly on the client side possible. Since tiled vector data can contain features, their attributes can be presented to the user when clicked, and a different style can be applied to highlight that selected feature.

NOTE	Even if the *OGC API - Maps - Part 1* does not provide a direct *GetFeatureInfo* equivalent, there is a strong tradition of *GetFeatureInfo* implementations that suggests a possible *OGC API - Maps* Standard future "part" could reintroduce a *GetFeatureInfo* equivalent - if users and implementers demand this capability.
NOTE	The second most commonly expected function, querying or filtering by the attributes of the features shown in the map, was never introduced in WMS. The same *OGC API - Maps* future "part" could provide the ability to filter by attributes using a CQL2 expression.

Annex D: Revision History

Date	Release	Editor	Primary clauses modified	Description
2019-03-21	0.0	C. Heazel	all	initial template
2019-06-28	T15-ER-0.00	J. Masó	all	initial version as work for the OGC API Hackathon in London
2019-09-09	T15-ER-0.01	J. Masó	all	Last version in the old document name with D014 and D16 together
2019-10-01	T15-ER-0.1	J. Masó	all	first version with the correct ToC as D014
2019-10-14	T15-ER-0.8	J. Masó	all	ready for OGC review
2019-10-21	T15-ER-0.9	J. Masó	all	results of the OGC review
2019-10-25	T15-ER-0.98	J. Masó	all	document ready for the 3 week rule in the Toulouse TC.
2019-12-09	T15-ER-0.99	C. Reed	various	Minor edits to Abstract, Exec Summary, and various other clauses prior to publication as a Public ER
2020-01-02	T15-ER-1.0	J. Masó	various	Final staff review and edits of Testbed-15 engineering report (OGC 19-069).
2023-04-20	0.8	J. Masó, J. St-Louis	all	Re-organized into several new requirements classes providing additional capability and flexibility.
2023-05-02	1.0.0rc1	J. Masó, J. St-Louis	various	First version of OGC API - Maps submitted to the OAB for review.
2023-12-22	1.0.0rc2	J. Masó, J. St-Louis	various	Implemented feedback from the OAB. Applied Carl Reed's suggestions. Added examples annex. Improved conformance and overview sections. Complete transition to https URIs. Added security considerations section. Clarified scaling and subsetting requirement classes. Added JPEG XL requirement class. Several other improvements. Second version of OGC API - Maps submitted to the OAB for review.

Annex E: Bibliography

- W3C/OGC: Spatial Data on the Web Best Practices, W3C Working Group Note 28 September 2017, https://www.w3.org/TR/sdw-bp/
- W3C: Data on the Web Best Practices, W3C Recommendation 31 January 2017, https://www.w3.org/TR/dwbp/
- W3C: Data Catalog Vocabulary, W3C Recommendation 16 January 2014, https://www.w3.org/TR/vocab-dcat/
- Internet Assigned Numbers Authority (IANA). **Link Relation Types** [online, viewed 2020-07-09], https://www.iana.org/assignments/link-relations/link-relations.xml
- OGC: OGC 06-042, OpenGIS Web Map Service (WMS) Implementation Specification 1.3.0 (2006) https://portal.opengeospatial.org/files/?artifact_id=14416
- OGC: OGC 19-069, OGC Testbed-15: Maps and Tiles API Engineering Report (2020) https://docs.ogc.org/per/19-069.html