OGC Web API Guidelines

Open Geospatial Consortium

Submission Date: YYYY-MM-DD

Approval Date: YYYY-MM-DD

Publication Date: YYYY-MM-DD

External identifier of this OGC® document: http://www.opengis.net/doc/{doc-type}/{standard}/{m.n}

Internal reference number of this OGC® document: YY-YYYY

Version: {m.n}

Category: OGC® Policy

Editor: Joan Maso, Chuck Heazel, Andreas Matheus, Frank Terpstra

OGC Web API Guidelines

Document type:	OGC® Policy
Document subtype:	
Document stage:	Draft
Document language:	English

Table of Contents

i. Abstract

This document is a **draft** policy of the OGC Architecture Board (OAB). The document defines a series of guidelines for Web Application Programming Interface (API) standards developed by the OGC. The OGC Web API Guidelines consist of a list of Design Principles. The policy document therefore applies to the OGC API suite of standards.

ii. Keywords

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

ogcdoc, OGC document, policy, OGC API, web API

iii. Preface

This document defines a series of policy requirements for Web API standards developed by the OGC.

iv. Submitting organizations

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

Organization name(s)

- Heazel Technologies
- UAB-CREAF
- Secure Dimensions
- Geonovum

v. Submitters

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

Table 1. Contacts

Name	**Affiliation**
Chuck Heazel	Heazel Technologies
Joan Maso	UAB-CREAF
Andreas Matheus	Secure Dimensions
Frank Terpstra	Geonovum

Chapter 1. Scope

This policy document presents a set of common design principles for developing Web APIs.

The design of common elements of OGC Web APIs should follow a common pattern to ensure easy adoption.

There are common design principles in main-stream IT that should be adopted to ease the adoption of OGC Web APIs.

But still, there are some aspects that would need to be agreed upon to ensure seamless APIs for different thematic topics in OGC.

In particular avoid that APIs are fundamentally different to access, process and manage different kinds of geospatial resources such as Features, Maps, Tiles, Coverages, Observations, Processes, etc.

Chapter 2. Conformance

This policy defines a series of policy requirements for OGC API standards.

Conformance with this policy shall be checked manually through review, or through automation where possible.

A checklist is provided for each Standards Working Group (SWG) to document how it complies with the guidelines and if any exemptions to the guidelines apply.

Completed checklists are to be reviewed by the OAB in order for candidate standards to be considered for Public Requests for Comment (RFC).

Chapter 3. References

OGC: OGC 05-020r25, Technical Committee Policies and Procedures, http://docs.opengeospatial.org/pol/05-020r25/05-020r25.html (2017)

OGC: OGC 09-046r5, OGC Naming Authority – Procedures, http://www.opengeospatial.org/standards/na (2018)

OGC: OGC 06-030r12, OGC Architecture Board Policies and Procedures, https://portal.opengeospatial.org/files/?artifact_id=81222 (2018)

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), also known as the 'ModSpec'.

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

4.1. specification

document containing recommendations, requirements and conformance tests for those requirements (source: OGC 08-131r3)

NOTE This definition is included for completeness. See Clause 5.3 of The Specification Model — A Standard for Modular specifications (OGC 08-131r3). This does not restrict what else a standard may contain, as long as it does contain the three types of element cited.

4.2. standard

specification that has been approved by a legitimate Standards Body (source: OGC 08-131r3)

NOTE This definition is included for completeness. Standard and specification can apply to the same document. While specification is always valid, standard only applies after the adoption of the document by a legitimate standards organization. The legitimate Standards Bodies for OGC consist of OGC, ISO and any of the other standards bodies accepted and used as a source of normative references by OGC or ISO in their standards. In the normal meaning of the word "standard", there are other conditions that may be required, but this standard has chosen to ignore them in the process of abstraction.

4.3. Web API

API using an architectural style that is founded on the technologies of the Web (source: DWBP)

NOTE Best Practice 24: Use Web Standards as the foundation of APIs in the W3C Data on the Web Best Practices provides more detail.

Chapter 5. Conventions

This document uses the normative terms (SHALL, SHOULD, etc) defined in Subclause 5.3 of [OGC 06-121r3], 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 comply with this specification.

Chapter 6. Guidelines

6.1. Purpose and Process

The implementation of Web APIs that allow the management, processing and use of geospatial information should be possible by anyone familiar with the Web APIs designed for main stream IT. However, when designing a Web API by multiple domain experts, and not only by one team, and when trying to address multi purpose usage, it becomes challenging to ensure a common design pattern among all teams.

To ensure that (i) Web API design across all different domains of expertise is coherent and (ii) the maximum from main stream IT design pattern is reused, the OGC Architecture Board (OAB) requested the elaboration of these guidelines. This process is done under the Architecture DWG with the collaboration of the OWS Common SWG. For the moment this is a living document to inspire additional discussion and refinement within and among DWG and SWG teams, and contribute our learnings and suggestions to the technological community at large. The final aim is to reach consensus and converge in a document that the OGC can approve as a Policy Document.

Even though functioning as a Policy Document, the use is more like a checklist to streamline the design and the review process in OAB and OGC Technical committee. The assessment from verifying the Web API design against the checklist should be submitted with the Web API draft standard to the OAB. It is possible to not follow all principles, but reasons for deviation should be given.

For the moment you should consider this to be a living, evolving document. Please create or comment on existing Issues to discuss changes, corrections, and enhancements to the principles.

6.2. Starting point

The starting point of the Design Principles listed here was taken from a presentation on OGC Web API Design Principles requires OGC portal login given during the OGC TC meetings in Orleans and Fort Collins. The presentation summerizes a collection of the Web API design principles used today by major players in main stream IT business. The purpose of the presentation is to ensure that the "common part of an API" is designed such that it can be re-used and a adopted easily. However, the initial presentation was not perfect in the sense that it might be incomplete and that there is room left for a good consensus discussion.

The original author of the presentation (Andreas Matheus), in collaboration with Charles Heazel, agreed to make the content available in this open GitHub repo for the purpose of creating a starting point in discussion and deriving a set of guidelines that could eventually be used to test OGC Web API Implementation Standards for conformance.

6.3. Design Principles

6.3.1. Principle #1 – Don’t Reinvent

For aspects and functional capabilities that are already solved in main-stream IT and meet geospatial community requirements, simply adopt these API elements.

Focus instead on geo-centric and domain specific requirements to create new APIs or extend existing APIs.

6.3.2. Principle #2 – Keep It Simple and Intuitive

Make the developer of the API successful as quickly as possible!

6.3.3. Principle #3 - Use Well-Known Resource Types

Identify your resource types and reuse existing definitions from the OGC Naming Authority resource type register (to be established).

Encodings of resource types should be associated with an IANA registered media type.

6.3.4. Principle #4 – Construct consistent URIs

Great Web APIs look like they were designed by a single team. The most obvious properties of an API are the access paths and the URL templates which define them. Therefore, OGC conventions for the construction of access path templates are essential. Some of these templates are emerging though the Web Feature Service 3.0 efforts. Before creating a new URI scheme, you should follow and build on existing approaches in OGC. If creating your own URI scheme, please explain your URI pattern.

Your API URI pattern should be documented, formalized, explained.

One existing approach in OGC is the following (simplified):

For resource types that consist of a collection of resources, the pattern at the end of the URI path is as follows where `resourceType` is in plural:

```
.../{resourceType}/{resourceId}
```

Where resources are nested, the path elements may be concatenated. For example:

```
.../collections - returns the list of feature collections
.../collections/highways - returns representation of the collection 'highways'
.../collections/highways/items - returns the features in the collection 'highways'
.../collections/highways/items/A8 - returns the feature 'A9' in the collection
'highways'
```

For resource types that consist of a single resource, the pattern at the end of the URI path is as follows where `resourceType` is in singular:

```
.../{resourceType}
```

For example:

```
.../collections/highways/schema - returns the schema for the features in the
collection 'highways'
.../collections/highways/metadata - returns the information about the features in the
collection 'highways'
```

Note that it doesn’t matter if you use singular or plural for your nouns to build the paths, but use a consistent pattern throughout your API!

6.3.5. Principle #5 – Use HTTP Methods consistent with RFC 2616

Include in your API design the use of all HTTP methods that operate on resources: **GET, POST, PUT, DELETE**

Define the semantics carefully when a method is invoked on a particular URI addressing a resource. E.g.

Resource	POST	GET	PUT	DELETE
../collections/highways/items	create a new highway	list all highways	bulk update of highways	delete all highways
… /collections/highways/items/A8	Error!	show A8	If exists: Update A8 else: Create A8	delete highway A8

Do not force all semantics in just HTTP GET!

Also consider support for other HTTP methods:

- HEAD to return HTTP Headers with no payload
- OPTIONS to support W3C CORS
- PATCH to update parts of an existing resource

6.3.6. Principle #6 – Put Selection Criteria behind the ‘?’

The radical idea behind the '?' concept is that everything **left** of the **'?'** (the path design) identifies a resource and that everything **right** of the **'?'** may select specific representaion(s) of parts or the entire resource.

For example:

```
.../collections/highways/items?id=A8 => returns highway A8
```

```
.../collections/highways/items?id=A8,A9 => returns highways A8 and A9
```

```
.../collections/highways/items/A8?time=2019-02-12T12:00:00Z => returns a highway A8
represetation at the given time
```

If you define a query parameter on a resource, define the API behaviour in all cases including error situations. For example, if you support an `id` parameter on a feature resource then define the semantics of the following query examples:

```
.../collections/highways/items/A8?id=A8 => should the request return *true* or *the
resource itself*?
```

```
.../collections/highways/items/A8?id=A81 => should the request return *false* or
'*NULL*' (assuming the id of A8 is not A81)?
```

Another example for a query parameter could be `properties` (which can also be combined with other parameters):

```
.../collections/highways/items/A8?properties=name,geometry => return the highway A8,
but only the name and geometry attributes
```

```
.../collections/highways/items/A8?time=2019-02-12T12:00:00Z&properties=name,geometry
=> return the highway A8 at the given time, but only the name and geometry attributes
```

Use of the query string to select resources is highly resource specific and must be described on a case by case basis.

6.3.7. Principle #7 – Error Handling and use of HTTP Status Codes

Note: Error Codes are the developers insight into your API. So be precise and as detailed as possible. Error handling is often one of the biggest complaints when using an API.

Associate each error situation of your API with the appropriate HTTP status code (see also Principle #8).

However, you may also consider supporting a "switch off" that always returns a status code 200 plus additional (debug / insight) information in the HTTP response body

```
e.g. ?suppress_response_codes=true
```

Return detailed human readable error no. + description
information on how to fix things + contact details

```
    { "developer_message": "…",
      "user_message": "...",
      "error_code": "...",
      "contact_details": "..."
    }
```

6.3.8. Principle #8 – Use of HTTP Status Codes

More then 70 HTTP status codes exist (summary in RFC 7231). You should reduce the use in your API to the most important ones. For example:

Option Set #1 – Basic set	Option Set #2 – Additional
- 100 - Continue	
- 200 - OK	- 201 - Created
	- 204 - No Content
	- 304 - Not Modified
- 400 - Bad Request	- 401 Unauthorized
	- 403 - Forbidden
	- 404 - Not Found
	- 405 - Method Not Allowed
	- 406 - Not Acceptable
	- 409 - Conflict
	- 410 - Gone
	- 412 - Precondition Failed
	- 415 - Unsupported Media Type
	- 422 - Unprocessable Entity
	- 429 - Too Many Requests
- 500 - Internal Server Error	- 503 - Service Unavailable

Be explicit which 30x status code your API supports. For any supported 30x follow the HTTP semantics.

6.3.9. Principle #9 – Use of HTTP Header

Define all HTTP Headers that your API supports.

Use HTTP Headers as intended by RFC 2616, but design your API to allow overwriting of HTTP Headers based on URL query parameters.

For support of caching consider to support entity tags and the associated headers. However, their use might be in conflict when implementing security requirements. For these cases, you should explicitly name those headers that must be overwritten to avoid caching.

6.3.10. Principle #10 - Content Negotiation

Content negotiation is an important, but special case of Principle #9.

Use HTTP request header like 'Accept' or 'Accept-Language' to request the response in a particular content type or language as defined in RFC 2616.

Use registered IANA Media Types whenever possible.

An example for content negotiation based on HTTP headers and with query parameter override:

```
HTTP 1.1 GET .../collections/highways/items/A8
accept: application/geo+json
=> should return the response using the GeoJSON encoding
```

```
HTTP 1.1 GET .../collections/highways/items/A8?accept=application%2Fgml%2Bxml
accept: application/json
=> should return the response using the GML encoding
```

6.3.11. Principle #11 - Pagination

APIs for large data collections should support pagination.

For example, use **limit** and **offset** as "query-string" parameters.

```
.../collections/highways/items?limit=50&offset=101 => returns upto 50 highways
starting at position 101
```

The API should return metadata with each response providing the total number of resources available (e.g. total) in the payload as well as the link to the next page.

As a supplement consider support for Web Linking (RFC 5988)

– Use HTTP Response Header to provide URLs for fetching the next / previous page – This approach is application neutral and should be provided by the API as the default

6.3.12. Principle #12 – Processing Resources

Use **verbs** to offer **operations** on resources, for example:

```
.../transform => represents a processing resource that allows to transform another
resource
```

The parameters of the process are provided as query parameters, for example:

```
.../transform?in=.../collections/highways/items/A8&toCRS=http://www.opengis.net/def/cr
s/EPSG/0/3258 => returns the A8 highway in the coordinate reference system ETRS89
lat/long
```

Note that the result of the example above may result in the same response as a selection/negotiation parameter on the resource (see Principle #6), for example:

```
.../collections/highways/items/A8?crs=http://www.opengis.net/def/crs/EPSG/0/3258
```

APIs may decide to offer processing resources as separate operations to support an explicit separation and highlight the processing capability. This allows to publish explicit metadata about the process, e.g., the input and output data structures.

6.3.13. Principle #13 – Support Metadata

This part of the API helps the developer to understand how to use data or processing resources. Two approaches exist how to achieve this:

(1) Start the URL path with 'metadata' to indicate that subsequent path identifies a resource for which the metadata is returned.

```
.../metadata/collections/highways/items/A8
```

(2) End the URL path with 'metadata' to indicate that the metadata is an integral part of the resource that can be fetched separately.

```
.../collections/highways/items/A8/metadata
```

Regardless of the approach taken, use it consistently.

You may use of the ‘?’ operator to send selection criteria (see Principle #6).

6.3.14. Principle #14 – Consider your Security needs

Try to follow common practices for security in Web APIs, for example:

- Host your API on HTTPS.
- Include support for authentication for the beginning.
- Consider consistent support for CRUD (Create, Read, Update, Delete) from the beginning (see Principle #5);
- support for Execute may be provided on processing resources (see Principle #12) or using POST (see Principle #5).

6.3.15. Principle #15 – API Description

Describing the API in human and machine readable form has value to the developer. Currently OpenAPI version 3 is common practice.

6.3.16. Principle #16 - Use IANA well-known identifiers

IANA and other standardization organizations have defined so called well known identifiers for different purposes. For example:

- Media types: https://www.iana.org/assignments/media-types/media-types.xhtml
- Link relations: https://www.iana.org/assignments/link-relations/link-relations.xhtml
- Well-known URIs: https://www.iana.org/assignments/well-known-uris/well-known-uris.xhtml

For example is it possible to differentiate between XACML or GeoXACML policies. XACML policies would be returned with the 'application/xacml+xml' media type and GeoXACML policies with media type 'application/geoxacml+xml'

6.3.17. Principle #17 - Use explicit geospatial relations

In many cases it is appropriate to use typed relation to explicitly declare links among resources. A special case are topological spatial relations between resources (e.g., contains, within, etc.) which are easy to derive with a GIS, but not with Web clients unless the relations are explicitly represented. The relations may either be explicitly included in the resource representation or in Link headers in the HTTP response header (see RFC 5988).

6.3.18. Principle #18 - Support W3C Cross-Origin Resource Sharing

If your Web API is accessed by Web-applications executed in a Web Browser, support W3C CORS (https://www.w3.org/TR/cors/). This allows to overcome the security restrictions introduced by the Same-Origin Policy (https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy) applied by the Web Browser to JavaScript based applications when trying to access your Web API.

In cross origin cases, as identified in W3C CORS, the HTTP request carries specific HTTP headers and it is expected by the Web Browser that associated HTTP response headers exist in the response. Otherwise the processing stops.

6.3.19. Principle #19 - Resource encodings

The API should provide resource representations based on the expectations of the developers.

You also have to decide whether or not the Web API should support a default encoding that every implementation has to support. You should recommend to support JSON and HTML as encodings for all resources. JSON is recommended as it is a commonly used format that is simple to understand and well supported by tools and software libraries; HTML is recommended as it is the standard encoding for Web content.

Still, the XML encoding should be supported as it is often required to meet specific security requirements. Also, many existing standards and OGC encodings are based on XML.

6.3.20. Principle #20 - Good APIs are testable from the beginning

Any OGC Web API developed according to these guidelines can be tested at design phase already. Considering all design principles including the identification of resource types, the effect of applying HTTP methods to them, the potential HTTP status codes, etc. provides the basis for documenting and implementing compliance tests in parallel to the API design.

Annex A: OGC Web API Guidelines

Each Standards Working Group (SWG) shall complete this checklist when the SWG submits a standard to the OAB for consideration ahead of the Public Request for Comment (RFC).

#	Principle	If and How the principle is met by the candidate standard
1	Don’t Reinvent	
2	Keep It Simple and Intuitive	
3	Use Well-Known Resource Types	
4	Construct consistent URIs	
5	Use HTTP Methods consistent with RFC 2616	
6	Put Selection Criteria behind the ‘?’	
7	Error Handling and use of HTTP Status Codes	
8	Use of HTTP Status Codes	
9	Use of HTTP Header	
10	Content Negotiation	
11	Pagination	
12	Processing Resources	
13	Support Metadata	
14	Consider your Security needs	
15	API Description	
16	Use IANA well-known identifiers	

17	Use explicit geospatial relations	
18	Support W3C Cross-Origin Resource Sharing	
19	Resource encodings	
20	Good APIs are testable from the beginning	

Annex B: Revision History

Date	Release	Editor	Primary clauses modified	Description
2018-07-09	0.0.1	Chuck Heazel	all	Initial text
2018-07-27	0.0.2	Andreas Matheus	multiple	
2018-08-02	0.0.3	Joan Maso	multiple	
2019-02-12	0.0.4	Andreas Matheus	multiple	