Open Geospatial Consortium

Publication Date: 2016-01-18

Approval Date: 2015-09-17

Posted Date: 2015-08-21

Reference number of this document: OGC 15-052r1

Reference URL for this document: http://www.opengis.net/doc/PER/tb11_rest_er

Category: Public Engineering Report

Editor: Frédéric Houbie

OGC® Testbed 11 REST Interface Engineering Report

Copyright © 2016 Open Geospatial Consortium.
To obtain additional rights of use, visit http://www.opengeospatial.org/legal/.

Warning

This document is not an OGC Standard. This document presents a discussion of technology issues considered in an initiative of the OGC Interoperability Program. This document does not represent an official position of the OGC. It is subject to change without notice and may not be referred to as an OGC Standard. However, the discussions in this document could very well lead to the definition of an OGC Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	Public Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
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.

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

Copyright © 2016 Open Geospatial Consortium.

Contents

Page

Copyright © 2016 Open Geospatial Consortium.

Figures Page

Tables Page

Copyright © 2016 Open Geospatial Consortium.

Abstract

REST architectural principles are associated with optimal functioning of the Web but their manifestation in geospatial Web services standards is not straightforward. This OGC Engineering Report (ER) examines their use both in existing OGC Services standards and in new or revised service standard proposals, some of which were implemented during OGC Testbed 11. The ER then defines possible uniform practices for developing bindings or interaction styles for OGC Web services that appropriately leverage REST principles.

Business Value

Web service standards that are easier to implement and more compatible with Web technologies will have a higher rate of implementation and a greater value for vendors, customers, and the OGC. The REST architectural style has the potential for simplicity, scalability, and resilience if it can be adapted to the effective exchange of geospatial information across the Web.

Keywords

ogcdoc, testbed 11, rest, ap, resource, http, web service, engineering report

OGC® Testbed 11 REST Interface Engineering Report

1 Introduction

1.1 Scope

This OGC Engineering Report (ER) is a deliverable of the OGC Testbed 11 activity. The ER describes the results of study of REST work at OGC. The ER also describes some guidelines for future definition of REST API within OGC process.

1.2 Document contributor contact points

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

Name	Organization
Frédéric Houbie (editor)	Luciad
Satish Sankaran	Esri
Joshua Lieberman	Tumbling Walls
Peter Vretanos	CubeWerx
Joan Masó	UAB-CREAF

1.3 Future work

The following items were identified for consideration in future initiatives:

- Uniform REST binding(s) for OGC services

REST API to support linked geospatial data

1.4 Foreword

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

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

Copyright © 2016 Open Geospatial Consortium.

2 Terms and definitions

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

2.1

Application Programming Interface (API)

An interface definition that permits invoking services from application programs without knowing details of their internal implementation.

3 Abbreviated terms

EPSG	European Petroleum Survey Group
GIS	Geographic Information System
GML	Geography Markup Language
ISO	International Organization for Standardization
OGC	Open Geospatial Consortium
SQL	Structured Query Language
SRS	Spatial Reference System
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

4 OGC Testbed 11 REST ER – Overview

One of the topics addressed by the OGC Testbed 11 Cross Community Interoperability (CCI) thread is the role of REST principles in OGC standards. Numerous discussions and attempts have been made to identify this role, determine how a RESTful suite of OGC service standards would work, and characterize the benefits it might bring. Within the Testbed, a number of participants implemented more RESTful versions of standard OGC services. This Engineering Report has two goals. One is to evaluate the current suite of OGC standards and policy statements with regard to REST principles. The other is to outline prospects for defining RESTful interfaces or bindings for OGC services, based on trial implementations in Testbed 11 and elsewhere, that maximize the benefits of this architectural style while retaining the service functionality on which client applications depend.

The following chapters document the results of the work conducted in OGC Testbed 11 for these tasks.

5 REST

5.1 What is REST ? [1]

REST (Representational State Transfer) is a term coined by Roy Fielding[2] in his doctoral dissertation to describe an architectural style for "distributed hypermedia systems" such as the Worldwide Web that are to have desirable characteristics including separation of concerns, scalability, resiliency, visibility, and reliability. This style rests on five constraints on how computing systems are configured and component interactions are carried out:

5.1.1 Client-Server

The uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.

5.1.2 Stateless

REST is an acronym for Representational State Transfer, so why statelessness? Although both clients and servers necessarily save their own persistent state information, the interactions that cause each to change state are themselves stateless. No interaction depends for its success on the content of other interactions. The necessary information to process a request, for example, is contained within the request itself, whether as part of the URI, query-string parameters, body, or headers.

Most of us who have been in the industry for a while are accustomed to programming within a container that provides "session state" across multiple HTTP requests. In REST, the client must include all information for the server to fulfil the request, resending state information or references as necessary if it applies to multiple requests. Statelessness enables greater scalability since the server does not have to maintain, update or

[1] Mainly extracted from http://www.restapitutorial.com/, one of the most concise and understandable REST explanations found during the analysis

[2] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

communicate that session state. Additionally, load balancers don't have to worry about session affinity for stateless systems.

So what's the difference between state and resource? Resource state is the information or capability a server uses to create resource representations – the data stored in the database, for instance. Application or client state is the information that affects application flow or user interactions in a client and is built up from (among other sources) resource representations exchanged with servers in stateless interactions.

5.1.3 Cacheable

As on the World Wide Web, clients can cache responses. Server responses implicitly or explicitly define themselves as cacheable, or not, to prevent clients reusing old or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, improving scalability and performance, as long as the requests, e.g. URL's are defined in such a way as to support as much reuse as possible. For example, if a client is able to construct many different requests for the same response, this prevents optimal caching performance.

5.1.4 Uniform Interface

The uniform interface constraint applies to interactions between client and server components, minimizing the special knowledge that each must have of the other and allowing maximum variety in the components that are able to interact with one another. It simplifies and decouples the architecture, which enables each part to evolve independently. As applied to the Worldwide Web, this uniform interface is generally taken to be based on proper and consistent use of HTTP, although other uniform interfaces are certainly possible, particularly for interaction styles not well supported by HTTP.

REST principles expand on the notion of a uniform protocol such as HTTP and describe several practices that increase the uniformity of the entire client-server interaction:

5.1.5 **Resource-Based**

In what is termed "resource oriented architecture (ROA), a server's capabilities are defined in terms of conceptual resources that are identified by URI's and addressable only by a uniform set of operators such as the HTTP verbs. By contrast, a Service Oriented Architecture (SOA) approach defines server capability in terms of specific, even idiosyncratic operations. It is generally true that a client application needs to be coded specifically for each specific server operation it invokes. Therefore, a small and uniform number of operators should make client development easier and increase the range of servers that a client can interact with. This presumes, of course, that how a client processes the representations of resources that a server returns is also somehow uniform or at least does not require specific, specialized code.

5.1.6 **Manipulation of Resources through Representations**

The resources themselves are conceptual (as far as the client is concerned) and any of a number of representations are what is actually exchanged with clients. For example, a server does not send its database, but rather, a HTML, XML or JSON document that encodes requested database records. Such documents may be expressed, for instance, in French and encoded in UTF-8, depending what the client requests and the server can provide. When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.

5.1.7 **Self-descriptive Messages**

Each representation message returned by a server is supposed to include enough information that a client is able to determine how to process the message without out-of-band code or information that is specific to that representation. An existing parser may be invoked, for example, by an Internet media type (previously known as a MIME type). While this is a valuable constraint for a resilient Web, the general effectiveness of media types in providing guidance for client processing has been somewhat overwhelmed by the variety of message types that have developed. Typically, a client application requires specialized code to deal with specialized media types or the ways in which specific representations may differ from general media types. This code is either built in based on out-of-band knowledge or it comes along with the response. Even HTML as a general media type is frequently supplemented by javascript code for specific application processing.

A notable form of self-descriptive message is, of course, hypermedia, in which various information media (text, pictures, videos, …maps) support application navigation by way of links. Where the processor of the document is a human (or the document is highly structured) and application flow consists of choosing the link to follow next, this type of self-descriptive message can be very effective.

Self-description also applies to responses explicitly indicating their cache-ability.

5.1.8 **Hypermedia as the Engine of Application State (HATEOAS)**

The exact interpretation of this practice has often been debated, but it clearly connects the choice of links to follow in hypermedia response documents with the flow and state of client applications. Changes in application state may be solely, primarily, or partly the result of following a hypermedia link to a new server interaction. The effect of this practice, however, is to transfer some or all of the logic of an application from client code to hypermedia representations returned from servers. This constraint further supports the resilient concept of a generic client that is able to interact with any server providing recognizable hypermedia responses and present to its user the options determined by the server. HATEPAS presupposes, however, that virtually all application processing is more appropriately carried out on a server than on a client

Copyright © 2016 Open Geospatial Consortium.

5.1.9 Layered System

Layers provide information hiding by presenting each server as simple component whether or not there is physically a population of components doing the actual processing. A client cannot ordinarily tell whether it is connected directly to the initiating server of a work pipeline or hierarchy or intermediary along the way. This increases separation of concerns and therefore resiliency in case opaque server workflow needs to change. Intermediary servers may also improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

5.1.10 Code-on-demand

Fielding's thesis actually includes a sixth architectural constraint that is considered optional, but carries to a logical conclusion that the less clients need to be equipped themselves for a specific application, the more loosely coupled a client server system can be. Clients in systems constrained this way are responsible for little more than executing code on documents that are both provided by the servers they interact with. Web browsers executing downloaded javascript functions may be considered an example of this constraint.

5.2 Contrasting REST with OGC Web Services

Careful consideration of published REST principles, constraints, and practices show how they can lead to a Web-like distributed computing system with admirable qualities of scalability and loose coupling. Two of goals identified with these qualities, however, are at some odds with both the concept of Web services and existing OGC services ecosystem. One goal is to abstract the model of server capabilities that a client needs to interact with, by means of a resource-representation approach and uniform interface. This leads to a common view that a uniform interface service standard such as HTTP is all that is required in the way of standards to promote interoperability between clients and servers. This has led directly to a proliferation of REST API specifications that are customized as to both server resources and client applications, with a corresponding drop in reusability as developers concentrate on the interactions needed for each project. It is generally thought that RESTful API's are easier to develop clients for. However, this is "ease" being offset by the need to continually re-develop on the client side as new improved API's are devised. This contrasts with the OGC approach of standardizing reusable service interfaces that can be utilized to access many different resources of specific types for the benefit of a range of applications.

There is a second REST goal of minimizing specific client knowledge of server capability, resource type, or representation format. Again, there is a range of interpretations of the advisability and importance of this goal. Some have even suggested that any REST API requiring more application capability in a client than parsing a few hypermedia types and following links (something close to a browser) should not be termed REST or RESTful. Regardless of terminology, this presents a clear view that an optimal separation between client and server components places most application logic on the server and not on the client. This is again a contrast with the distributed architectures for geospatial information processing targeted by OGC Web services. The role of these services has been to provide distinct, reusable portions of backend computing capability such as data access or transformation that are then assembled by clients into complete applications (whether or not they are directly human-facing).

When those client applications are browser-based, though, they are often (although not always) supported by an additional service tier of Web-based API's that implement portions of processing logic specific to the particular application. This tier sometimes even comprises the actual client that interacts with OGC services such as Web Feature Service instances and then passes the results on to the browser-based user interface. These application services are rarely standardized per se, although there are communications and/or widget frameworks that provide considerable regularity to client development tasks and may yet be a standardization target. Creating such application services would be much more difficult, though, if the components for rendering a map or searching a feature collection or storing large-scale datasets had to be re-created for each application.

The conclusion drawn by most Testbed 11 participants appears to be that whatever name is used (RITHMIC – Resource Interaction Through HTTP Method Invocation – was proposed as a substitute), there is value specifically in the geospatial domain in attempting to strike a balance between the value of specific REST principles and practices, and the demonstrated value of standardized component Web services .

5.3 Common REST API Practices to Consider

Whether their use can be termed RESTful or not (according to the constraints mentioned previously), a few recommended REST-like suggestions have been collected here. These six relatively easy to implement software practices often result in better, more usable services.

5.3.1 Use HTTP Verbs as more than a transport protocol

API consumers are capable of sending GET, POST, PUT, and DELETE verbs, and these verbs greatly enhance the clarity of what a given request does. Also, GET requests must not change any underlying resource data. Measurements and tracking may still occur, which updates data, but not resource data identified by the URI.

Generally, the four primary HTTP verbs are used as follows.

GET: Read a specific resource (by an identifier) or a collection of resources.
The HTTP GET method is used to read (or retrieve) a representation of a resource. In the "happy" (or non-error) path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

Copyright © 2016 Open Geospatial Consortium.

According to the design of the HTTP specification, GET (along with HEAD) requests are used only to read data and not change it. Therefore, when used this way, they are considered safe. That is, they can be called without risk of data modification or corruption—calling it once has the same effect as calling it 10 times, or none at all. Additionally, GET (and HEAD) should be idempotent, which means that making multiple identical requests ends up having the same result as a single request.

Do not expose unsafe operations via GET—it should never modify any resources on the server.

PUT: Update a specific resource (by an identifier) or a collection of resources. Can also be used to create a specific resource if the resource identifier is known before-hand.

PUT is most-often utilized for update capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.

However, PUT can also be used to create a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. Again, the request body contains a resource representation.

Alternatively, use POST to create new resources and provide the client-defined ID in the body representation—presumably to a URI that doesn't include the ID of the resource (see POST below).

On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. A body in the response is optional—providing one consumes more bandwidth. It is not necessary to return a link via a Location header in the creation case since the client already set the resource ID.

PUT is not a safe operation, in that it modifies (or creates) state on the server, but it is idempotent. In other words, if you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call.

If, for instance, calling PUT on a resource increments a counter within the resource, the call is no longer idempotent. Sometimes that happens and it may be enough to document that the call is not idempotent. However, it's recommended to keep PUT requests idempotent. It is strongly recommended to use POST for non-idempotent requests.

DELETE: Remove/delete a specific resource by an identifier.

DELETE is pretty easy to understand. DELETE is used to delete a resource identified by a URI.

On successful deletion, return HTTP status 200 (OK) along with a response body, perhaps the representation of the deleted item (often demands too much bandwidth), or a wrapped response. Either that or return HTTP status 204 (NO CONTENT) with no response body.

HTTP-spec-wise, DELETE operations are idempotent. If you DELETE a resource, it's removed. Repeatedly calling DELETE on that resource ends up the same: the resource is gone. If calling DELETE say, decrements a counter (within the resource), the DELETE call is no longer idempotent. As mentioned previously, usage statistics and measurements may be updated while still considering the service idempotent as long as no resource data is changed. Using POST for non-idempotent resource requests is recommended.

There is a caveat about DELETE idempotence, however. Calling DELETE on a resource a second time will often return a 404 (NOT FOUND) since it was already removed and therefore is no longer findable. This, by some opinions, makes DELETE operations no longer idempotent, however, the end-state of the resource is the same. Returning a 404 is acceptable and communicates accurately the status of the call.

POST: Create a new resource. Also a catch-all verb for operations that don't fit into the other categories.

The POST verb is most-often utilized to **create** new resources. In particular, it's used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource. In other words, when creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an ID (new resource URI), etc.

On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.

POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Making two identical POST requests will most-likely result in two resources containing the same information.

5.3.2 Provide Sensible Resource Names

Producing a great API is 80% art and 20% science. Creating a URL hierarchy representing sensible resources is the artistry. Sensible resource names (which are just URL paths, such as /customers/12345/orders) improve the clarity of what a given request does. Appropriate resource names provide context for a service request, increasing understandability of the API. Resources are viewed hierarchically via their URI names, offering consumers a friendly, easily-understood hierarchy of resources to leverage in their applications. The resource structure in itself itself makes each resource easier for a client to understand and interact with.

Here are some quick-hit rules for URL path (resource name) design.

Copyright © 2016 Open Geospatial Consortium.

- Use identifiers in your URLs instead of in the query-string. Using URL query-string parameters is fantastic for filtering, but not for resource names.
  - Good: /users/12345
  - Poor: /api?type=user&id=23
- Leverage the hierarchical nature of the URL to imply structure.
- Design for your clients, not for your data.
- Resource names should be nouns. Avoid verbs as resource names. It makes things more clear. Use the HTTP methods to specify the verb portion of the request.
- Use plurals in URL segments to keep your API URIs consistent across all HTTP methods, using the collection metaphor.
  - Recommended: /customers/33245/orders/8769/lineitems/1
  - Not: /customer/33245/order/8769/lineitem/1
- Avoid using collection verbiage in URLs. For example 'customer_list' as a resource. Use pluralization to indicate the collection metaphor (e.g. customers vs. customer_list).
- Use lower-case in URL segments, separating words with underscores ('_') or hyphens ('-'). Some servers ignore case so it's best to be clear.
- Keep URLs as short as possible, with as few segments as makes sense.

Look at some widely used APIs to get the hang of this and leverage the intuition to refine API resource URIs. Some example APIs are:

Twitter: https://dev.twitter.com/rest/public

Facebook: https://developers.facebook.com/docs/graph-api/reference

LinkedIn: https://developer.linkedin.com/docs/rest-api

More in the Geo domain, Google APIs has some good hints for integration of spatial/temporal characteristics in the URLs :
https://developers.google.com/maps/documentation/webservices/

These examples emphasize identifying each granular resource in the URL path, even for dimensional parameters such as tile or raster coordinates and reserving query parameters for other purposes (filtering, sorting, …).

5.3.3 Use HTTP Response Codes to Indicate Status

Response status codes are essential for making the best use of the HTTP specification. There are quite a number of them to address the most common situations. In the spirit of having our RESTful services embrace the HTTP specification, our HTTP-based services should return relevant and meaningful HTTP status codes. For example, when a resource is successfully created (e.g. from a POST request), the API should return HTTP status code 201. A list of valid HTTP status codes is available here which lists detailed descriptions of each.

Suggested usages for the “Top 10” HTTP Response Status Codes are as follows.

200 OK

General success status code. This is the most common code. Used to indicate success.

201 CREATED

Successful creation occurred (via either POST or PUT). Set the Location header to contain a link to the newly-created resource (on POST). Response body content may or may not be present.

204 NO CONTENT

Indicates success but nothing is in the response body, often used for DELETE and PUT operations.

400 BAD REQUEST

General error for when fulfilling the request would cause an invalid state. Domain validation errors, missing data, etc. are some examples.

401 UNAUTHORIZED

Error code response for missing or invalid authentication token.

403 FORBIDDEN

Error code for when the user is not authorized to perform the operation or the resource is unavailable for some reason (e.g. time constraints, etc.).

404 NOT FOUND

Used when the requested resource is not found, whether it doesn’t exist or if there was a 401 or 403 that, for security reasons, the service wants to mask.

405 METHOD NOT ALLOWED

Used to indicate that the requested URL exists, but the requested HTTP method is not applicable. For example, POST /users/12345 where the API doesn’t support creation of resources this way (with a provided ID). The Allow HTTP header must be set when returning a 405 to indicate the HTTP methods that are supported. In the previous case, the header would look like “Allow: GET, PUT, DELETE.”

409 CONFLICT

Copyright © 2016 Open Geospatial Consortium.

Whenever a resource conflict would be caused by fulfilling the request. Duplicate entries, such as trying to create two customers with the same information, and deleting root objects when cascade-delete is not supported are a couple of examples.

500 INTERNAL SERVER ERROR

Never return this intentionally. The general catch-all error when the server-side throws an exception. Use this only for errors that the consumer cannot address from their end.

For a full list of HTTP status codes, see IETF page[3].

5.3.4 Create Fine-Grained Resources

When starting out, it's best to create APIs that mimic the underlying application domain. However, it's much easier to create larger resources later from collections of individual resources than it is to create fine-grained or individual resources from larger aggregates.

5.3.5 Consider Connectedness

One of the principles of REST is connectedness via hypermedia. While services are still useful without them, APIs become more self-descriptive and discoverable when links are returned in the response. At the very least, a 'self' link reference informs clients how the data was or can be retrieved. Additionally, utilize the HTTP Location header to contain a link on resource creation via POST (or PUT). For collections returned in a response that support pagination, 'first', 'last', 'next' and 'prev' links at a minimum are very helpful.

Regarding linking formats, there are many. The HTTP Web Linking Specification (RFC5988[4]) explains a link as follows:

a link is a typed connection between two resources that are identified by Internationalized Resource Identifiers (IRIs) [RFC3987[5]], and is comprised of:

- *A context IRI,*
- *a link relation type*
- *a target IRI, and*
- *optionally, target attributes.*

A link can be viewed as a statement of the form "{context IRI} has a {relation type} resource at {target IRI}, which has {target attributes}."

3 http://www.ietf.org/assignments/http-status-codes/http-status-codes.xml

4 http://tools.ietf.org/search/rfc5988

5 http://tools.ietf.org/search/rfc3987

Another often used solution is to place links in the HTTP Link header[6] as recommended in the specification, or embrace a JSON representation of this HTTP link style (such as Atom-style links, see: RFC4287[7]) in your JSON representations.

Link: <https://www.example.com/api/v1/cars?offset=15&limit=5>; rel="next",<https://www.example.com/api/v1/cars?offset=50&limit=3>; rel="last",<https://www.example.com/api/v1/cars?offset=0&limit=5>; rel="first",<https://www.example.com/api/v1/cars?offset=5&limit=5>; rel="prev"

HTTP Header can be used to return additional information like the total count of entries, like X-Total-Count.

5.4 ROA vs SOA

A critical consequence of adopting a uniform service interface approach such as with HTTP is that design challenges shift from designing new operations to configuring new resources. This shift in thinking has been described as moving from Service Oriented Architecture (SOA) to Resource Oriented Architecture (ROA). A ROA design approach has advantages for resilient distributed computing by hiding many details of remote processing, but is not a trivial shift in thinking.

5.4.1 Service Oriented Architecture (SOA)

SOA is based on the concept of a **service**. Depending on the service design approach taken, each SOA service is designed to perform one or more activities by implementing one or more service operations. As a result, each service is built as a discrete piece of code. This makes it possible to reuse the code in different ways throughout the application by changing only the way an individual service interoperates with other services that make up the application, versus making code changes to the service itself. SOA design principles are used during software development and integration.

SOA generally provides a way for consumers of services, such as web-based applications, to be aware of available SOA-based services. For example, several disparate departments within a company may develop and deploy SOA services in different implementation languages; their respective clients will benefit from a well-defined interface to access them.

SOA defines how to integrate widely disparate applications for a Web-based environment and uses multiple implementation platforms. Rather than defining an API, SOA defines the interface in terms of protocols and functionality. An endpoint is the entry point for such a SOA implementation.

[6] http://www.w3.org/wiki/LinkHeader

[7] http://tools.ietf.org/search/rfc4287#section-4.2.7

Copyright © 2016 Open Geospatial Consortium.

5.4.2 Resource Oriented Architecture (ROA)

ROA is a specific set of guidelines of an implementation of the REST-style architecture. According Leonard Richardson and Sam Ruby in their book entitled 'RESTful Web Services.'

ROAs are based on four concepts.

- Resources (e.g., the article about REST in the Wikipedia). Their names (URIs).
- The URI is the name and – dereferenced to a URL – is the address of a resource. For example,http://www.wikipedia.org/wiki/Representational_State_Transfer.
- Resource representations. A resource is the concept or template from which representations are created.
- Links between resources. Ideally a hypermedia representation of a resource contains links to other resources.

And four properties.

- **Addressability**. Addressable applications expose a URI for every piece of information they might conceivably serve.
- **Statelessness**. Statelessness means that every HTTP request happens in complete isolation. The server never relies on information from previous requests.
- **Connectedness**. A Web service is connected to the extent that you can put the service in different states just by following links and filling out forms.
- **A uniform interface**. In Web ROAs, HTTP is the uniform interface. GET method to retrieve a representation of a resource, PUT method to a new URI or POST method to an existing URI to create a new resource, PUT method to an existing URI to modify a resource and DELETE method to remove an existing resource. Probably HTTP methods are not a perfect interface but what is important is the uniformity. The point is not that GET is the best name for a read operation, but that GET means "read" across the Web. Given a URI of a resource, everybody knows that to retrieve the resource s/he has to send a GET request to that URI.

Since a SOA definition is independent of the technical architecture of the services, it encompasses all REST/HTTP applications. ROA can be seen as a term to describe that part of a SOA implemented following the guidelines stated before. That is, ROA is less general than SOA since it is not independent of the technical architecture of the services. The term ROA is often used to emphasize that such an architecture in based on HTTP objects that respond to one or more of the standard HTTP methods. Why? Because SOA have been traditionally focused on interfaces and when people talk about interfaces they tend to use terms like “method”, “operation”, etc. which are strongly related to the RPC-style. Thus, to avoid misunderstandings the term ROA is used to make clear that we are talking about REST-style architectures.

6 The History of OGC and REST

For years, REST has been a hot topic of deliberations in the OGC community. The need for embracing REST into the OGC standards model is evident. The table below contains most of the OGC documents (Presentations, Best Practices, ERs, etc.) dealing with REST. The first document about REST was written during the OWS Testbed Phase 5 for the SWE domain. Phase 6 also had some active work on the REST subject. A list of OGC documents dealing with REST can be found in Annex A.

Although this subject has been discussed for a long time, there are not many OGC documents explaining how to implement a REST API. Some OGC documents will be described in the following sections.

A quick analysis of the existing OGC standards shows that there is still some work to be done to align to the REST tips described above.

Table 1 – Cross-reference of OGC standards to REST practices

	WMS	WFS	WCS	WPS	SOS	SPS	CSW	WMTS
Use HTTP methods explicitly.	Y	N	Y*	N	N	N	N	Y
Be stateless.	Y	Y	Y	Y	Y	Y	Y	Y
Expose directory structure-like URIs.	N	N	N	N	N	N	N	Y
Use HTTP Error codes	N	N	N	N	N	N	N	N
Transfer XML, JavaScript Object Notation (JSON), or image.	Image	XML	Any	Any	XML	XML	XML	Image

6.1 Analysis of REST Initiatives within OGC

6.1.1 Geoservices REST – Overview

The GeoServices REST Specification provides a standard way for web clients to communicate with geographic information system (GIS) servers through Representational State Transfer (REST) technology. Clients issue requests to the server through structured URLs. The server responds with map images, text-based geographic information, or other resources that satisfy the request. Although the GeoServices REST Specification was originally built to communicate with Esri’s ArcGIS® Server product, the specification has been opened such that developers can expose the GeoServices REST Specification request structure from other back-end GIS servers or processes. This has

Copyright © 2016 Open Geospatial Consortium.

allowed its implementation by others outside the Esri platform. The specification was then worked processed as a candidate OGC standard in the GeoServices SWG. The draft version of the candidate is available. The candidate standard specifies commonly used resources in an implementation of the GeoServices REST API as well as extensibility requirements.

The GeoServices REST Specification describes a catalog of web services that are designed for different GIS functions (map, geocode, and so on). The GeoServices REST API specification document has been structurally broken down into multiple documents. An implementer interested in implementing feature services, would only need to understand the core, the catalog spec and the Feature service spec to fully implement the GeoServices Feature Service. The Core and the catalog spec provides the overall services viewpoint, while the feature services doc (part 4) talks specifically to the feature service related functionality.

Part 1 : Core

Part 2 : Catalog

Part 3 : Map Service

Part 4: Feature Service

Part 5: Geometry Service

Part 6: Image Service

Part 7: Geoprocessing Service

Part 8: Geocoding Service

6.1.2 Why implement the GeoServices REST specification?

The GeoServices REST Specification offers a simple way for applications to request map, feature, attribute, and image information from a GIS server. Developers who adopt the GeoServices REST Specification are choosing a proven implementation that has been widely deployed and exercised in the field and that exposes server-side resources to a broad range of clients and applications. They are also choosing a JSON-based, RESTful specification that will make the server instantly usable by thousands of developers working in popular client-side development environments with the ArcGIS web mapping APIs for JavaScript™, Flex™, Silverlight®, iOS®, and Android™, all of which are powered by the GeoServices REST Specification. GeoServices REST technology is already widely implemented by US and other Government and military agencies and by many companies and open source users around the world. The API does not need to be dependent on any particular company’s underlying products or data structures.

6.1.3 How to implement the GeoServices REST specification.

To implement the GeoServices REST Specification, developers architect the back-end server to respond to specifically structured REST requests in an expected way. For example, if someone issues a request to the server to export a map image, such as http://<mapservice-url>/export?bbox=-127.8,15.4,-63.5,60.5, the server should return a map image with a lower left coordinate of (-127.8, 15.4) and an upper right coordinate of (-63.5, 60.5). How the server generated the image is not as important as the fact that it responded in an expected way when issued a URL whose structure followed the GeoServices REST Specification.

The full GeoServices REST Specification is described in the OGC candidate standard documents. Developers can choose how much or how little to implement. For example, if geocoding or geoprocessing operations are not exposed by the GIS server, developers may not need to implement the Geocode Service or GP Service piece of the candidate standard.

All resources and operations exposed by the GeoServices REST Specification are accessible through a hierarchy of endpoints or uniform resource locators (URLs) for each available GIS service. When using the GeoServices REST Specification, users typically start from a well-known endpoint, which represents the server catalog. From the catalog, different types of resources are available as child nodes. These resources comprise services for mapping, geocoding, and so on.

The GeoServices REST Specification is stateless because REST does not keep track of transactions from one request to the next. Each request must contain all the information necessary for successful processing.

6.1.4 Resources and Operations

The GeoServices REST Specification works with a hierarchy of resources. Each service type recognized by the GeoServices REST Specification (map, geocode, and so on) is a resource and has a unique URL. Although a REST system always returns only representations of resources to client, for the sake of simplicity, the resources of the GeoServices REST Specification are divided into two types: resources and operations (also called controller resources).

6.1.5 Response Formats

Many resources in the GeoServices REST Specification have a parameter, f, that denotes the response format. Developers can program resources to respond to REST requests with various data formats, including JSON, HTML, and KMZ. Current implementations of the spec by Esri also support GeoJSON as an additional JSON based format.

At the least, the JSON response format should be implemented, and examples for doing so are provided in the specification. Other formats are optional, and they can be exposed through the f parameter; however, formats other than JSON are not detailed in the specification.

6.1.6 REST and Pragmatic Considerations

The GeoServices REST specification is based on both RESTful principles and pragmatic considerations. These considerations include – support for various aspects of the HTTP protocol in commonly used environments like JavaScript, Adobe Flex or Microsoft Silverlight, in commonly used web browsers, and in proxys. Many of these environments do not typically offer complete support for all of the HTTP standard. This lead the initiators of the GeoServices REST specification to make some choices. Here are some of the pragmatic considerations.

- Certain web browsers lack support for HTTP PUT and DELETE (except using scripting methods).
- Some rich internet application clients do not fully support PUT and DELETE.
- From a practical implementation standpoint from within the browser for HTML applications one may want to use HTTP GETs via dynamic script tags whenever this can be done safely. Dynamic script tags for GET operations are cacheable and do not force clients on different domains to go through a proxy.
- HTTP POST is needed whenever the size of the URL may be longer than 2048 characters. This isn't usually a factor for most API designs, but in the context of geographic information it happens quite frequently (serialized geometries can easily be larger than 2000 characters).
- Often firewalls and proxies strip out HTTP PUTs and DELETEs. This can be mitigated by forcing SSL for all requests, but this is not practical. Some RESTful APIs recommend HTTP method overloading to get around this, which would be a hack. In this case one would use POST, but in the header (or in the query parameter) one specifies that one really wants to do a PUT or DELETE.
- Cross domain scripting needs have been mostly solved using proxy servers and support for JSONP (JSON with padding). These make it impossible to support all the HTTP error codes. As discussed in Clause 8, most responses with JSON content will use an HTTP status code 200 and the JSON content will either be a resource representation, the result of a controller resource operation or an exception.
- MIME types are advertised using query parameters in the URL (e.g., "?f=json") rather than using the HTTP headers. This too can be attributed to the fact that many times proxy servers tend to strip out header information and hence a more practical/safer approach of using parameters in the URL has been used for this purpose.

6.1.7 Summary

As part of the GeoServices REST API SWG process, there were many issues that were seriously considered. A good review of these comments will provide an expanded view of a) REST as a pattern for geospatial web services, b) Pragmatic considerations that are necessary for creating "real world" implementations using REST and c) the synergies that exist between the GeoServices model and the ISO/OGC Baseline.

Some sample URL's from Esri's implementation of the open GeoServices REST specification are provided below. These are rich examples for readers of the ER to review and play with to better understand the value of the GeoServices REST specification. As the OGC document is still a draft, the sample implementations below are not "fully compliant" implementations of the OGC candidate standard.

http://sampleserver1.arcgisonline.com/ArcGIS/rest/services

http://sampleserver2.arcgisonline.com/ArcGIS/rest/services

http://sampleserver3.arcgisonline.com/ArcGIS/rest/services

http://sampleserver4.arcgisonline.com/ArcGIS/rest/services

http://sampleserver5.arcgisonline.com/ArcGIS/rest/services

http://sampleserver6.arcgisonline.com/ArcGIS/rest/services

The OGC SWG group for the GeoServices SWG is currently inactive. In the future the SWG may decide to restart its activities to take this process to completion; making the candidate standard a full OGC standard. Until then, the work that went into this effort holds many answers for the other OGC activities seeking to leverage RESTful patterns. Web environments today increasingly favor patterns that reflect the natural evolution of the web over rigid academic design arguments. The candidate standard finely balances the rigor necessary for standardization with the practical aspects of building a GIS experience that stays faithful to practical Web 2.0 principles.

6.1.8 Further References

Draft OGC document: GeoServices REST API – Part 1-8 (OGC -12-054r1)

GeoServices REST API – Relationship with the OGC standards baseline (OGC 12-062r2)

GeoServices REST API – JSON Schemas and Examples (OGC 12-068r2)

GeoServices REST API – RFC Comments (OGC 12-164)

GeoServices REST specification (published by Esri): http://www.esri.com/industries/landing-pages/geoservices/geoservices

Copyright © 2016 Open Geospatial Consortium.

6.2 WaterML 2.0 API

Very recently, the WaterML SWG presented a REST interface and JSON encoding for WaterML 2.0 (OGC 15-033 – OGC WaterML2.0 part 2 – RESTful API and JSON encoding) Best Practice document –. The document specifies a RESTful API and JSON encoding of WaterML2.0 part 2. This API and encoding was used in the part 2 Interoperability Experiment (IE) to test the information model. The implementation was found to be useful for the IE participants and thus has been documented as a best practice.

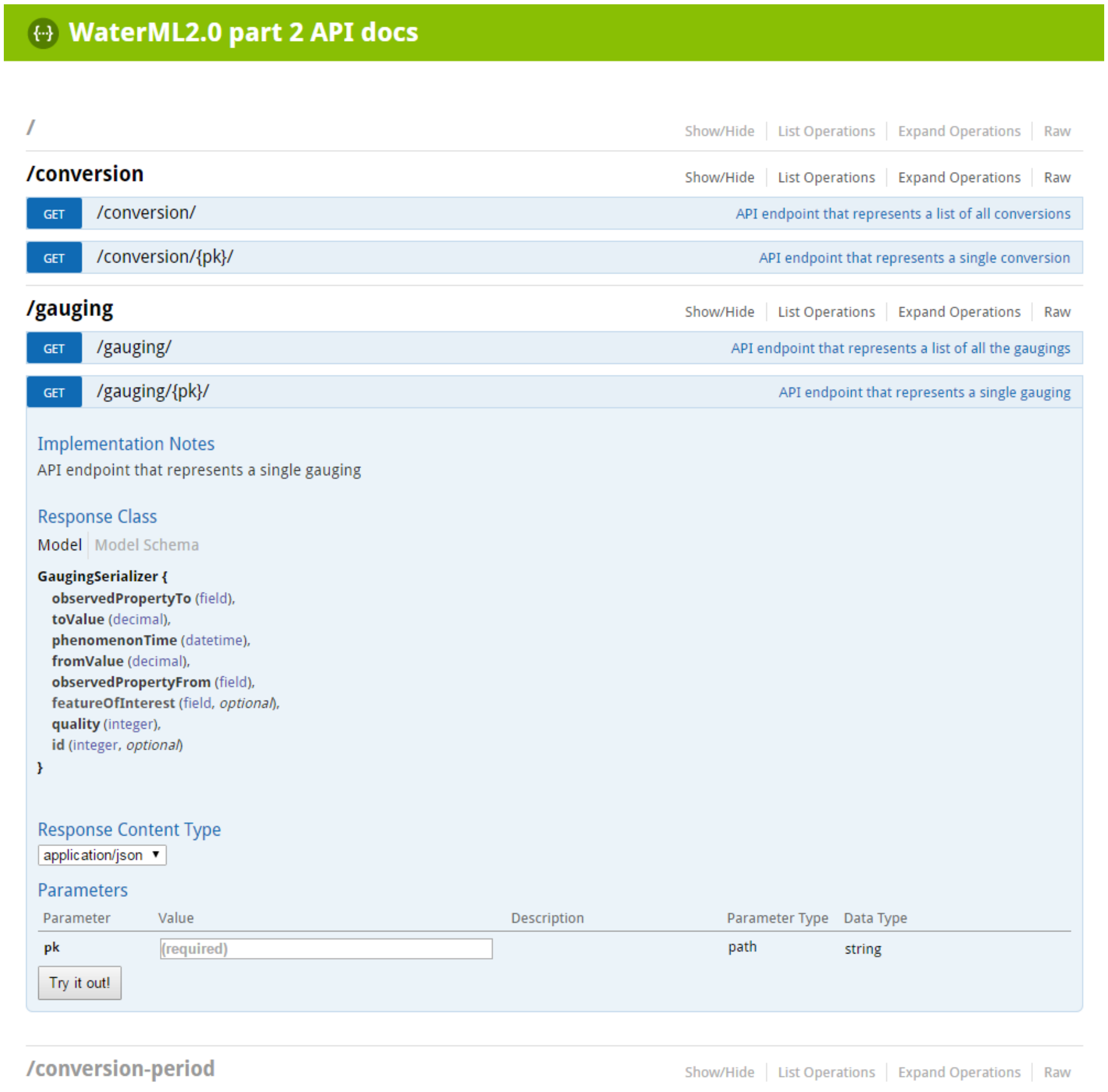
The RESTful API design was based on the requirements outlined by the IE participants and through an iterative implementation. The Best Practice scope is based on the WaterML2.0 information model. There are parts of the implementation that would require harmonization with any future OGC-wide RESTful and JSON practices.

In the case of the Best Practice document, it is a read-only service, so it only used GET. With the exception of plurals for resource in path, this API follows the guidelines described above.

The WaterML2 REST API (http://waterml2.csiro.au/rgs-api/v1/) is very simple as it defines the resources that can be managed at the root URL and it uses links extensively so that HATEOAS principle can be applied.

The available documentation is also very easy to understand and interactive. The most noticeable difference of this API compared to other OGC initiatives is that is starts from the Resources, which is the logical path for a REST API.

Figure 1 – WaterML2.0 part 2 API docs


WaterML2.0 part 2 API docs

/ Show/Hide | List Operations | Expand Operations | Raw

/conversion Show/Hide | List Operations | Expand Operations | Raw

GET /conversion/ API endpoint that represents a list of all conversions

GET /conversion/{pk}/ API endpoint that represents a single conversion

/gauging Show/Hide | List Operations | Expand Operations | Raw

GET /gauging/ API endpoint that represents a list of all the gaugings

GET /gauging/{pk}/ API endpoint that represents a single gauging

Implementation Notes

API endpoint that represents a single gauging

Response Class

Model | Model Schema

```
GaugingSerializer {
  observedPropertyTo (field),
  toValue (decimal),
  phenomenonTime (datetime),
  fromValue (decimal),
  observedPropertyFrom (field),
  featureOfInterest (field, optional),
  quality (integer),
  id (integer, optional)
}
```

Response Content Type

application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
pk	(required)		path	string

Try it out!

/conversion-period Show/Hide | List Operations | Expand Operations | Raw

6.3 Web Map Tile Service (WMTS) RESTful binding

One of the first OGC standards that adopted REST principles was WMTS. Inspired in the previous community standard called TMS, WMTS adopted some of the RESTful principles. To do so, WMTS defines 3 resource types: The ServiceMetadata document, the tile and the FeatureInfo report. Each resource type has its own MIME type. A WMTS service only accepts GET operations to retrieve resources. Creation, update or deleting of resources is not provided by the WMTS REST protocol binding and those operations are left open to the implementation (e.g. some implementations will provide a GUI interface based on HTTP while others will provide a command line set of instructions). The ServiceMetadata document is the single entry point to a service. Then tile resources can be retrieved (in the form of an image) and eventually, each pixel of each tile becomes a resource of the FeatureInfo type the representation of it is usually information about the features depicted in it in the form of a textual format.

Copyright © 2016 Open Geospatial Consortium.

One of the particularities of WMTS is that it uses the URL template standard to specify a URL pattern that all tile and FeatureInfo URL follows. The URL template standard extends the URL standard to allow to specify parameter names between '{'and'}' that will be processed by the client and substituted by actual values. This way, with a single URL template the client is able to create a URL of the needed tile (or a FeatureInfo) and retrieve a representation using a GET request. Even if this patter is also adopted in OpenSearch, the use of URL template has raised criticism.

6.3.1 Criticism of the URL template approach.

There are two criticisms of the URL template approach. The first is that URL templates are not opaque but follow some predefined structure. The second criticism is that this means of link navigation is not entirely consistent with HATEOAS principles.

In a completely RESTful HATEOAS implementation of a tile service, the service metadata document would point to a single tile that includes the whole extent provided by the layer. The tile format would be a text format that includes URL links to the tiles in the next level and the tile image linked or embedded (an alternative to this is use HTTP headers as specified in Clause 5. If the client requested a tile from the next zoom level, the response would provide new URL links to the spatially adjacent neighbors and the next higher and lower zoom levels. In this approach clients would have all the information to navigate through resources without previous knowledge of the tile URL pattern.

In practice, this has proven to result in remarkably awkward map applications. The client still needs to know crucial information about the individual tiles and tile hierarchy. The client needs to know something about the geospatial position of each tile. Otherwise, the client will not be able arrange the tiles adequately in the viewport and to cut out the unwanted regions of the marginal tiles. Crawling through a set of tiles in order to compute how to arrange them is just painful. Another factor to take into account is that sometimes users will require jumping from one zoom level to another that is far from the current one or will request a completely new location far from the previous view. In this case the status information that the server can communicate to the client in each tile in the form of new links becomes impractical. As a general principle, we think that the geospatial component introduces implicit geospatial relations between tiles that clients need to know in advance and exploit to provide a good user experience. These geospatial relations / templates cover the functionality that HATEOAS might provide step-by-step but with the necessary added efficiency of being able to request only the needed tiles out of possibly millions of possibilities.

6.3.2 Caching and considerations for future parameter extensions.

The WMTS standard does not make any assumption as to how or when tiles will be created in the server side. They can be pre-rendered or created on the fly. One way or another, the URL template is an effort to ensure that each tile is available with only one URL, making caching more efficient.

Some people argue that the KVP approach commonly used in OGC standards is also RESTful. This has some merit, but there are drawbacks. KVP is too flexible and allows for any order in the parameters. In addition, some standards allow for floating point numbers in KVP values associated to some keys (e.g. the BBOX). When combined this opens the door to an almost infinite number of URL variants for the same resource and makes efficient caching almost impossible.

The way WMTS is defined requires specifying all possible values of any new extra dimension in the service metadata document. The goal is to ensure that the values of these parameters will be written in the URLs in a single form with no ambiguity (the same provided by the service metadata document).

Recently, a WMTS time extension has been proposed. The proposers of the extension argue that listing all time values in the service metadata document is not convenient for long time series, and want to provide just an interval. This creates ambiguity in the way time has to be written in the URL depending on the precision used by the client. To prevent multiple URLs pointing to the same resource a format template is proposed. This is an extension of the URL template standard that will allow providing a format pattern. A example of what it has been proposed is: {time:YYYY/MM/DDThh:mm} where YYYY represents the year, MM represents the month, DD represents the day, hh represents the hour and mm represents the minutes (as used in some software such as MS Excel). In our opinion, this can be also used for other future parameter extensions such as elevation or temperature in the same way: {elevation:###00} or {temperature:##.#} where # represents a digit.

The need for having a single URL (or at least a small number) pointing to each resource needs to be taken into account in any WMTS extension.

6.3.3 WMTS Simple

The WMTS simple profile has been tested during the Testbed 11 activity. WMTS simple makes RESTful binding mandatory and imposes one of the two possible TileMatrixSets defined in the standard. A client using WMTS simple profile can ignore the Service Metadata document of a WMTS service conformant to this profile. Service discovery is made possible by communicating the URL template of a layer in the WMTS service directly by any communication means such as Facebook, Twitter, or an email. The following email fragment illustrates how this can be done.

```
Hey,
I have a WMTS 1.0.0 service up at “http://mycompany.com/wmts”.
It can be accessed with a standard WMTS client application.
It’s also compatible with the WMTS-Simple profile, with the
following URL templates:

“Oceans” layer:

http://mycompany.com/wmts/1.0.0/tiles/oceans/default/smerc/{TileM
atrix}/{TileCol}/{TileRow}.png
```

Copyright © 2016 Open Geospatial Consortium.

```
The "Transportation" layer (Rand-McNally style):

http://mycompany.com/wmts/1.0.0/tiles/transportation/RandMcNally/
smerc/{TileMatrix}/{TileCol}/{TileRow}.png
```

By implementing this profile, clients can more easily combine data coming from different services including from other WMTS instances and even from some tile implementations that are not OGC WMTS based, such as some current distributions of Open Street Map (OSM). In fact, most of these tiling services are implicitly following most of the WMTS requirements without saying it.

Even if the simple profile makes REST binding mandatory, it is questionable if it is more RESTFul than the previous version of WMTS. By imposing this profile, the client status description becomes simpler due to the TileMatrixSet si fixed. Nevertheless, clients do not exchange the status with services in HATEOAS way either.

6.4 REST binding for WFS 2.5

6.4.1 Introduction

The forthcoming WFS 2.5 specification includes a REST binding that, unlike existing OGC standards which are service-oriented, is a resource-oriented description of a web feature server that uses the standard HTTP interface as meaningfully as possible to manipulate and manage feature resources.

Testbed 11 provided an opportunity to test web feature services implementing the REST binding as described in the WFS 2.5 implementation standard. This clause summarizes the salient elements of the RESTful web features services used, primarily in the Urban-Climate Resilience thread, during the OGC Testbed 11 and described in OGC 11-080r1.

6.4.2 Basic service elements

6.4.2.1 Service root

For interactions with a RESTful WFS to commence a client needs to discover the service root of the WFS. The format of the service root URL is not specified in the WFS 2.5 standard and as such alleviates the need to define service and version negotiation protocols as those elements may (or not) be incorporated into the published service root URL in any manner that is meaningful to the service implementation.

The following table lists the service roots for the RESTful WFSs used in the OGC Testbed 11.

Table 2 – OGC Testbed 11 WFS 2.5 Implementations

Provider	Service root

CubeWerx	6.4.2.1.1.1.1.1 http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
Geomatys	http://ows11.geomatys.com/constellation/WS/wfs/MOZOSM/2.0.0
IBM	http://ogcwfs.mybluemix.net/wfs/2.5

A WFS's service metadata document (i.e. its capabilities document) (see OGC 06-121r3, OGC 11-080r1) is accessible at its service root.

6.4.2.2 Representation of features

The canonical representation of features, as defined in the WFS 2.5 standard, is GML 3.2 (see OGC 07-036). However, the standard allows other representation to be used as well and during Testbed 11 extensive testing was undertaken using a JSON, and specifically GeoJSON (see http://geojson.org/geojson-spec.html) representation of features.

6.4.2.3 Content negotiation

Content negotiation is the mechanism by which a client or user agent and a server arrive at a mutually agreeable representation of resources in the server's response. During Testbed 11, two approaches to content negotiation were tested.

The primary method of content negotiation used in the OGC Testbed 11 was the HTTP Accept header (see http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html). In a WFS request, the client simply sets the value of the Accept header to a list of one or more MIME types representing the client's preferred resources encodings. Each listed MIME type may also include a quality factor letting the server know the relative degree of preference of each listed MIME type. The following as examples of the Accept header indicating the client's preferences for a GeoJSON feature representation but also capable of accepting a GML representation:

```
Accept : application/vnd.geo+json, application/gml+xml ; version=3.2
```

In the absence of quality factors the MIME types are listed in preferred representation first.

6.4.2.4 Hypermedia controls

One of the main features of the RESTful architectural style is connectedness. That means that resources should link to each other and their representations and that from any state of the systems, links should be available to allow a client to navigate to a next valid state. The WFS 2.5 standard uses the ATOM link element (see OGC 07-147r2) to support the inclusion of hypermedia controls in the responses of the service.

Copyright © 2016 Open Geospatial Consortium.

The ATOM link element includes a “rel” attribute describing the relationship between the current document and the linked document. The value of the “rel” attribute can be a value from the IANA Link Relations registry (see RFC 5988) or a value that has been registered with the OGC naming authority.

The two primary places where hypermedia controls can appear in a WFS’s response are in the server’s capabilities document and in a query response.

Hypermedia controls in the server’s capabilities document allow a client to:

(a) Obtain the application schema that the service offers. This schema must be available as a GML 3.2 (see OGC 07-036) application schema but other schema representations are allowed.
(b) Navigate to a collection of features of this type.
(c) Obtains the schema of this feature type.
(d) Access (i.e. query) features of this feature type.

The following XML fragments is an example from the capabilities document of one of the servers using in Testbed 11:

```
   <FeatureType>
      <atom:link xmlns:atom=”http://www.w3.org/2005/Atom”
rel=”collection”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0
/ows11/wwAccess”/>
      <atom:link xmlns:atom=”http://www.w3.org/2005/Atom”
rel=”describedby” type=”application/schema+xml”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0
/ows11/wwAccess”/>
      <Name>cw:wwAccess</Name>
      <Title>wwAccess</Title>
      <DefaultCRS>urn:ogc:def:crs:EPSG::4326</DefaultCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::42110</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::3857</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::4267</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::4269</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::32758</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::32759</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::32760</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:OGC::CRS41001</OtherCRS>
      <ows:WGS84BoundingBox>
         <ows:LowerCorner>172.4565 -43.8190</ows:LowerCorner>
         <ows:UpperCorner>172.9755 -43.3975</ows:UpperCorner>
      </ows:WGS84BoundingBox>
   </FeatureType>
```

The fragment describes one to the features types, wwAccess, offered by the service and includes two hypermedia controls.

The first control, with rel=”collection”, will access features of this this type by performing a query on the server using the default values of the query parameters (see Table 4). In this case, the control is a link to a document that contains 10 features (the default value for the “count” parameter) of the “wwAccess” type.

The second control, with rel=”describedby”, accesses a document containing the schema of the feature type wwAccess.

In this way, a client (knowing the service root) can navigate to next, valid, states without necessarily having to be aware of the details of accessing the service. The only requirement is that the client recognizes the “rel” value and resolves the accompanying link.

Hypermedia controls in a query response may allow a client to:

(a) Navigate to the service (i.e. get its capabilities document) that offers the feature.
(b) Navigate to the collection that contains the feature.
(c) Navigate to alternative representations of the current feature

The following XML fragment shows a sample query response with hypermedia controls:

```
<wfs:member>
   <atom:link rel=”service”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
”/>
   <atom:link rel=”collection”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess”/>
   <atom:link rel=”alternate” type=”application/gml+xml; version=2.1”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fgml%2Bxml%3B%20version%3D2.1”/>
   <atom:link rel=”alternate” type=”application/gml+xml; version=3.1”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fgml%2Bxml%3B%20version%3D3.1”/>
   <atom:link rel=”alternate” type=”application/gml+xml; version=3.2”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fgml%2Bxml%3B%20version%3D3.2”/>
   <atom:link rel=”alternate” type=”application/x-bxfs+xml;
version=0.0.3”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fx-bxfs%2Bxml%3B%20version%3D%220.0.3%22”/>
   <atom:link rel=”alternate” type=”application/rss+xml”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Frss%2Bxml”/>
   <atom:link rel=”alternate” type=”application/vnd.google-
earth.kml+xml”
href=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
```

Copyright © 2016 Open Geospatial Consortium.

```
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fvnd.google-earth.kml%2Bxml"/>
   <atom:link rel="alternate" type="application/atom+xml"
href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fatom%2Bxml"/>
   <atom:link rel="alternate" type="text/html"
href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=te
xt%2Fhtml"/>
   <atom:link rel="alternate" type="application/vnd.geo+json"
href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fvnd.geo%2Bjson"/>
   <atom:link rel="alternate" type="application/vnd.shp+octet-stream"
href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=ap
plication%2Fvnd.shp%2Boctet-stream"/>
   <wwAccess
    gml:id="CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000">
      <Geometry>
         <gml:Point gml:id="GID1"
                    srsName="urn:ogc:def:crs:EPSG::4326">
         <gml:pos>-43.59451450970821 172.5609996851931</gml:pos>
         </gml:Point>
      </Geometry>
      <uid>WWMH-3044</uid>
      <AccessType>Standard Manhole</AccessType>
      <SourceId>2504S00145</SourceId>
      <LocationCe>Non Verified</LocationCe>
      <NearestPru>1</NearestPru>
      <HansenId>62682</HansenId>
      <n>5734677.7</n>
      <e>2474562.44</e>
      <WwAccessID>3044</WwAccessID>
      <Compkey>9521</Compkey>
      <HeightAbov>22.04</HeightAbov>
      <Constructi>1</Constructi>
      <Depth>1.1</Depth>
      <Maintenanc>City Water and Waste</Maintenanc>
      <ServiceSta>Abandoned</ServiceSta>
      <Decommissi>2013-04-30</Decommissi>
      <YearLaid>1971-01-01</YearLaid>
      <SAPInterna>IE000000000010739686</SAPInterna>
      <LastEditDa>2015-01-26</LastEditDa>
      <LastEditUs>CCITY\TredinnickC</LastEditUs>
      <WwAccessSt>3453</WwAccessSt>
      <Checked>0</Checked>
   </wwAccess>
</wfs:member>
```

This response fragment includes a hypermedia control that links the feature back to the service that offers it (i.e. rel="service"), a hypermedia control that links the feature back to the collection that contains it (i.e. rel="collection") and a set of hyperlinks that provide alternative representations of this feature including HTML, ATOM and KML representations.

6.4.2.5 URI Templates

One aspect of connectedness discussed in the Cross Community thread and briefly investigated during Testbed 11 is the use of URI templates as an alternative to hypermedia controls. As an example of this, Open Search uses URI templates to describe an interface by which a client can make requests for resources from the service.

Open Search defines an element named “Url” that has similar attributes to the ATOM link element. The following tables maps the attributes of the “Url” element to the attributes of the “atom:link” element.

Table 3 – Mapping Open Search Url attributes to ATOM link attributes

Open Search attribute name	*Description*	*ATOM link attribute name*
rel	The role of the resource being described in relation to the description document.	Rel
type	The MIME type of the resource being described.	Type
template	The URL template that can be used to form a URL through variable expansion.	Href

Open search defines its own variables and rules for template expansion that seem to be a subset of RFC 6570 – URI Template. It is anticipated that if the URI template approach were to be adopted in OGC, the full RFC-6570 syntax would be used.

In the capabilities fragment example in clause 7.4.2.4, hypermedia controls were used to point clients to the schema of the feature type as well as point the client the collection of features of that type. Similar capabilities can be performed using URI templates. Assuming the existence of a “Url” element in the WFS schema, the example can be recast as:

```
   <Url rel=”collection” type=”application/gml+xml; version=3.2”
template=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2
.5.0/ows11/{featureType}”/>
   <Url rel=”describedby” type=”application/schema+xml”
template=”http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2
.5.0/ows11/{featureType}”/>
```

Copyright © 2016 Open Geospatial Consortium.

```
.
.
.
<FeatureType>
   <Name>cw:wwAccess</Name>
   <Title>wwAccess</Title>
   <DefaultCRS>urn:ogc:def:crs:EPSG::4326</DefaultCRS>
   <OtherCRS>urn:ogc:def:crs:EPSG::42110</OtherCRS>
   <OtherCRS>urn:ogc:def:crs:EPSG::3857</OtherCRS>
   <OtherCRS>urn:ogc:def:crs:EPSG::4267</OtherCRS>
   <OtherCRS>urn:ogc:def:crs:EPSG::4269</OtherCRS>
   <OtherCRS>urn:ogc:def:crs:EPSG::32758</OtherCRS>
   <OtherCRS>urn:ogc:def:crs:EPSG::32759</OtherCRS>
   <OtherCRS>urn:ogc:def:crs:EPSG::32760</OtherCRS>
   <OtherCRS>urn:ogc:def:crs:OGC::CRS41001</OtherCRS>
   <ows:WGS84BoundingBox>
      <ows:LowerCorner>172.4565 -43.8190</ows:LowerCorner>
      <ows:UpperCorner>172.9755 -43.3975</ows:UpperCorner>
   </ows:WGS84BoundingBox>
</FeatureType>
```

Because URI templates define a general pattern for forming URLs to access the schema or collection of features, the templates need to exist outside any specific feature type description in the capabilities document – which is why the Url elements appears outside the FeatureType element in the example above.

The two approaches are similar; however, the template approach does not follow the principles of HATEOAS (see 6.1.1, https://en.wikipedia.org/wiki/HATEOAS) because the URLs are no longer opaque. Furthermore, the URI template approach places the extra burden on the client of knowing how to expand a template as per RFC-6570 whereas in the hypermedia approach the fully formed URL is available and the client simply needs to resolve it. The implications of this difference were graphically illustrated during Testbed 11 by the fact that in the hypermedia case a standard browser could be used to act as a very simple WFS client. This was not possible with the template approach because the browser is unaware of the RFC-6570 expansion rules and the substitution variables that the WFS would need to define in order to use URI templates.

During Testbed 11 it was further determined that there is no reason why the two approaches cannot co-exist within the same service metadata or response document. In this way clients that know how to handle URI templates can use the embedded templates whereas unaware clients can simply follow the hypermedia controls.

CHANGE REQUEST: post a change request that optionally allows a server to include URI templates in the capabilities document and query responses of a web feature service.

6.5 Summary of resources

The following table summarizes the WFS resources used and tested during Testbed 11.

Table 4 – Summary of WFS 2.5 REST resources used in OGC Testbed 11

Resource class	Description	Access path
Capabilities document	The complete service metadata document describing the service and the feature types it offers.	/
Schema	The complete application schema of the server.	6.5.1.1.1.1.1.1 {schema URL}[2]
Feature Type	A named collection of features of the same type.	{ftype URL}[3]
Feature	A member of a feature type's collection (i.e. a feature).	6.5.1.1.1.1.1.2 {feat URL}[4]
Feature Type Property	6.5.1.1.1.1.1.3 A property from the schema of a feature type.	6.5.1.1.1.1.1.4 {ftype URL}/{prop}[1]
Feature Property	6.5.1.1.1.1.1.5 A property from the schema of feature.	6.5.1.1.1.1.1.6 {feat URL}{prop}[1]

1. {prop} = The name of a feature property; it is appended to a URL before any query parameters.
2. {schema URL} = The URL to the application schema the service offers; it is specified at the first nesting level within the wfs:FeatureTypeList element in the server's capabilities document using an atom:link element with rel="describedby".
3. {ftype URL} = The URL to a collection of features of this type; it is specified at the first nesting level within the wfs:FeatureType element in the capabilities document using an atom:link element with rel="collection".
4. {feat URL} = The URL of a feature.

6.6 Feature access

6.6.1 Introduction

The REST binding of the WFS 2.5 standard defines two classes of feature access or query: simple queries and complex queries.

Simple queries are those that access features from a single homogenous collection.

Copyright © 2016 Open Geospatial Consortium.

Complex queries are those that access multiple feature types and/or include complex predicates such as joins (standard, temporal, spatial). Two classes of complex query are defined, ad hoc queries and stored queries.

Testbed 11 concentrated on simple queries since those are processed in the expected REST manner using the GET method to access the desired feature/feature type (i.e. resource) that is identified by its URL.

6.6.2 Query parameters

A resource URL can include query parameters. Query parameters may be used to control the output format and content of the response; control the resolution of referenced resources in the response; identify subsets of features base on predicates; sort the response feature.

The full set of query parameters that may be appended to a resource URL are defined in the following table. Those rows highlighted in green indicate that the parameter was actually tested and used in the course of Testbed 11.

Table 5 – WFS 2.5 query parameters

Parameter name	*Notes*
outputFormat	See the WFS standard.
6.6.2.1.1.1.1.1 f	Alias for outputFormat.
resultType	Valid value are: hits = only return the number of features in the result set **results** = return the result set contained within a wfs:FeatureCollection element cursor = return the first feature of the result set and include a link to the next feature in the result set, etc.
count	See the WFS standard.
startIndex	See the WFS standard.
Resolve	See the WFS standard.
resolveDepth	See the WFS standard.
resolveTimeout	See the WFS standard.
nameSpaces	See the WFS standard.
Lock	Valid values are: true, **false**, {lockid} “true” means that the server attempts to lock all the feature that

	the query identifies subject to the value of the lockAction parameter “false” means don’t try to lock anything {lockid} means: (a) for previously locked feature: refresh the lock expiry (b) for previously unlocked features: lock them with this lockId if possible (again, subject to the value of the lockAction parameter)
lockExpiry	In seconds, default is **300**
lockAction	Valid values are: ALL, **SOME**
releaseAction	Valid values are: ALL, **SOME**
sortBy	See the WFS standard.
propertyName	**6.6.2.1.1.1.1.2** See the FES standard.
Filter_language	See the FES standard.
Filter	See the FES standard.
featureId	Retrieves the feature with the specified id within a wfs:FeatureCollection response container.
6.6.2.1.1.1.1.3 bbox	See the FES standard.
6.6.2.1.1.1.1.4 geometry	WKT-encoded geometry
6.6.2.1.1.1.1.5 crs	CRS for “geometry”
spatialOp	One of: Equals, Disjoint, Touches, Within, Overlaps, Crosses, **Intersects**, Contains
time	ISO8601 time instance or interval
temporalOp	One of: After, Before, Begins, **During**, EndedBy, Ends, Tequals, Meets, MetBy

6.6.3 Examples

6.6.3.1 Introduction

The following examples were derived from the OGC Testbed 11activity and illustrate feature access using a standard browser, **Firefox** in this case, as a simple WFS client.

Copyright © 2016 Open Geospatial Consortium.

While reviewing the examples, it should be kept in mind that the default value for the Accept header for the Firefox browser is:

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/*;q=0.8*

thus indicating that the preferred output format is HTML. Other browsers may have other default values for the Accept header and so the behavior may be different than what is described in the text of this clause.

"This links in this clause are live and access one of the servers used during the OGC Testbed 11. If problems are encountered please contact pvretano[at]cubewerx.com. This server shall be maintained for the contractual obligation of 6 months from the end of June, 2015."

6.6.3.2 Accessing a services capabilities document

Accessing the service root accesses, the server's capabilities or metadata document:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11

Because the default value of the Accept header for Firefox requests HTML output, the response shall be formatted as HTML as shown in Figure 1.

6.6.3.3 Accessing the application schema of the service

Within the capabilities document is a link where *rel="describedby"*. Accessing this link will retrieve the server's application schema.

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/schema

The default output format is GML 3.2 (i.e. application/gml+xml; version=3.2) and so the schema is presented as a GML 3.2 application schema.

6.6.3.4 Accessing the feature collection

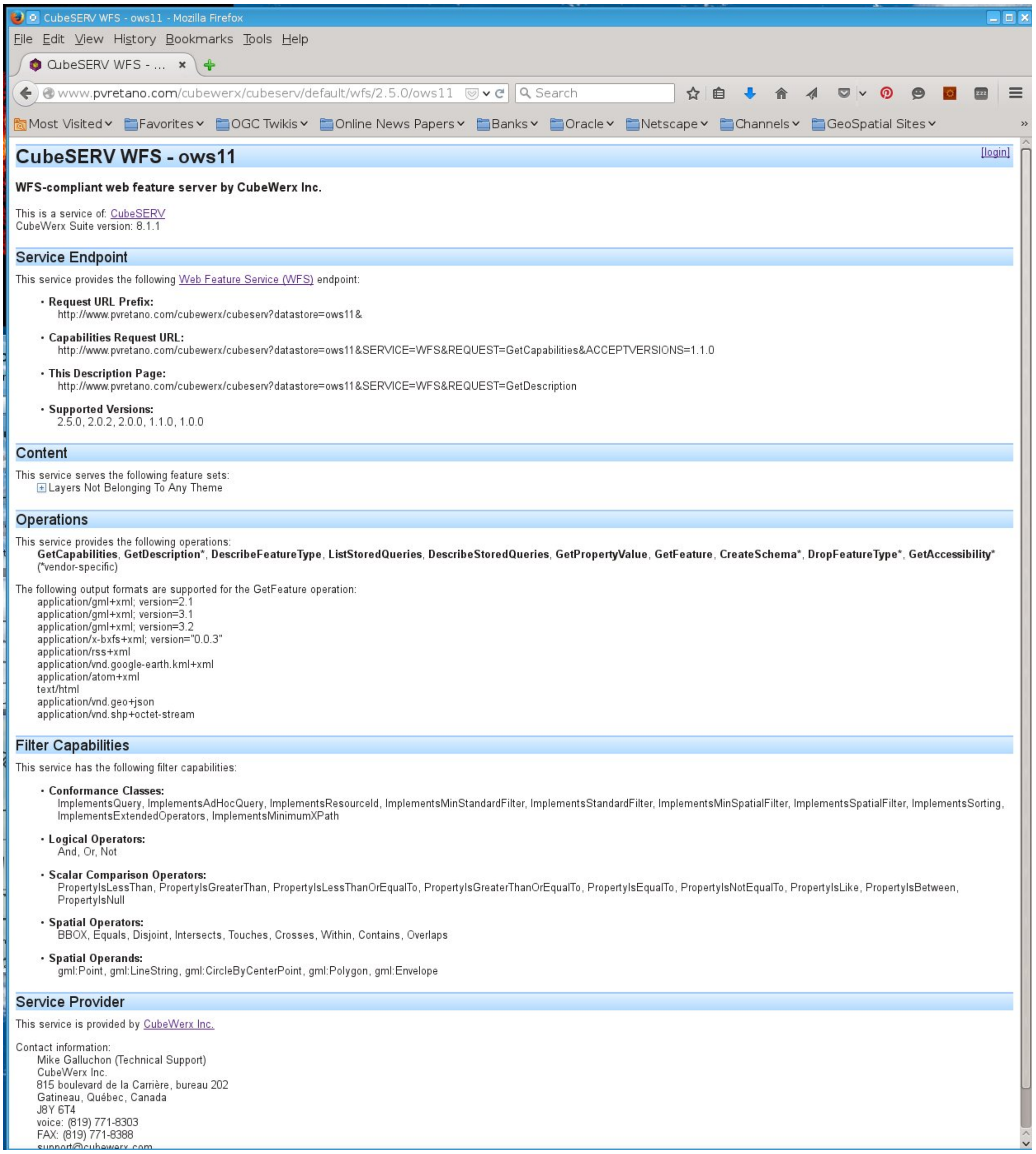
Each feature that the server offers is described in the server's capabilities document using the wfs:FeatureType element. That description includes a link where *rel="collection"*. Accessing that link will execute the default query which returns 10 features of that type. In this example we access the wwAccess feature type :

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess

If you view this link in your browser you will see an HTML representation of the wwAccess features because the preferred output format of the browser is HTML and the server is capable of generating HTML. The default output is not styled but including a style sheet, using a vendor extension "css", http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess?css=http://www.pvretano.com/cubewerx/css/ows11_table.css

makes the output look a little better.

Figure 2 – REST WFS Capabilities document in HTML


CubeSERV WFS - ows11 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
CubeSERV WFS - ...
www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
Search
Most Visited
Favorites
OGC Twikis
Online News Papers
Banks
Oracle
Netscape
Channels
GeoSpatial Sites
[login]

CubeSERV WFS - ows11

WFS-compliant web feature server by CubeWerx Inc.

This is a service of: CubeSERV
CubeWerx Suite version: 8.1.1

Service Endpoint

This service provides the following Web Feature Service (WFS) endpoint:

- **Request URL Prefix:**
  http://www.pvretano.com/cubewerx/cubeserv?datastore=ows11&
- **Capabilities Request URL:**
  http://www.pvretano.com/cubewerx/cubeserv?datastore=ows11&SERVICE=WFS&REQUEST=GetCapabilities&ACCEPTVERSIONS=1.1.0
- **This Description Page:**
  http://www.pvretano.com/cubewerx/cubeserv?datastore=ows11&SERVICE=WFS&REQUEST=GetDescription
- **Supported Versions:**
  2.5.0, 2.0.2, 2.0.0, 1.1.0, 1.0.0

Content

This service serves the following feature sets:
⊞ Layers Not Belonging To Any Theme

Operations

This service provides the following operations:
GetCapabilities, **GetDescription***, **DescribeFeatureType**, **ListStoredQueries**, **DescribeStoredQueries**, **GetPropertyValue**, **GetFeature**, **CreateSchema***, **DropFeatureType***, **GetAccessibility***
(*vendor-specific)

The following output formats are supported for the GetFeature operation:
application/gml+xml; version=2.1
application/gml+xml; version=3.1
application/gml+xml; version=3.2
application/x-bxfs+xml; version="0.0.3"
application/rss+xml
application/vnd.google-earth.kml+xml
application/atom+xml
text/html
application/vnd.geo+json
application/vnd.shp+octet-stream

Filter Capabilities

This service has the following filter capabilities:

- **Conformance Classes:**
  ImplementsQuery, ImplementsAdHocQuery, ImplementsResourceId, ImplementsMinStandardFilter, ImplementsStandardFilter, ImplementsMinSpatialFilter, ImplementsSpatialFilter, ImplementsSorting, ImplementsExtendedOperators, ImplementsMinimumXPath
- **Logical Operators:**
  And, Or, Not
- **Scalar Comparison Operators:**
  PropertyIsLessThan, PropertyIsGreaterThan, PropertyIsLessThanOrEqualTo, PropertyIsGreaterThanOrEqualTo, PropertyIsEqualTo, PropertyIsNotEqualTo, PropertyIsLike, PropertyIsBetween, PropertyIsNull
- **Spatial Operators:**
  BBOX, Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, Overlaps
- **Spatial Operands:**
  gml:Point, gml:LineString, gml:CircleByCenterPoint, gml:Polygon, gml:Envelope

Service Provider

This service is provided by CubeWerx Inc.

Contact information:
Mike Galluchon (Technical Support)
CubeWerx Inc.
815 boulevard de la Carrière, bureau 202
Gatineau, Québec, Canada
J8Y 6T4
voice: (819) 771-8303
FAX: (819) 771-8388
support@cubewerx.com

Copyright © 2016 Open Geospatial Consortium.

6.6.3.5 Accessing a specific feature in GML

Accessing a specific feature is simply a matter of resolving the feature's URL:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000

Once again, since we are using the browser, the output shall appear as HTML. However, this behavior can be overridden by using the "outputFormat" or "f" query parameters to request the response is a specific format. In this case:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?f=application/gml%2Bxml;%20version=3.2

the response will be in XML and in this case:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?f=application/vnd.geo%2Bjson

the response will be GeoJSON.

In both cases accessing a feature via its URL generates the bare features in a manner similar to that of performing a GetFeatureById operation in previous versions of the WFS. There is not container element; simply the base feature. Using the "featureId" parameter on the feature type or collection URL will retrieve the feature inside of a response container:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess?featureId=CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000

Notice now that the response is inside a wfs:FeatureCollection container and also contain numerous hypermedia controls.

6.6.3.6 Accessing a subset of features

As illustrated in clause 7.3.6.4, accessing a feature type without any query parameters returns a default set of 10 features. Other subsets may be retrieved by simply includes query parameters (see Table 4) on the collection's URL. For example, all the wwAccess features within a specific bounding box can be retrieved using the "bbox" query parameter:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess?bbox=-43.4380,172.6489,-43.3947,172.7108

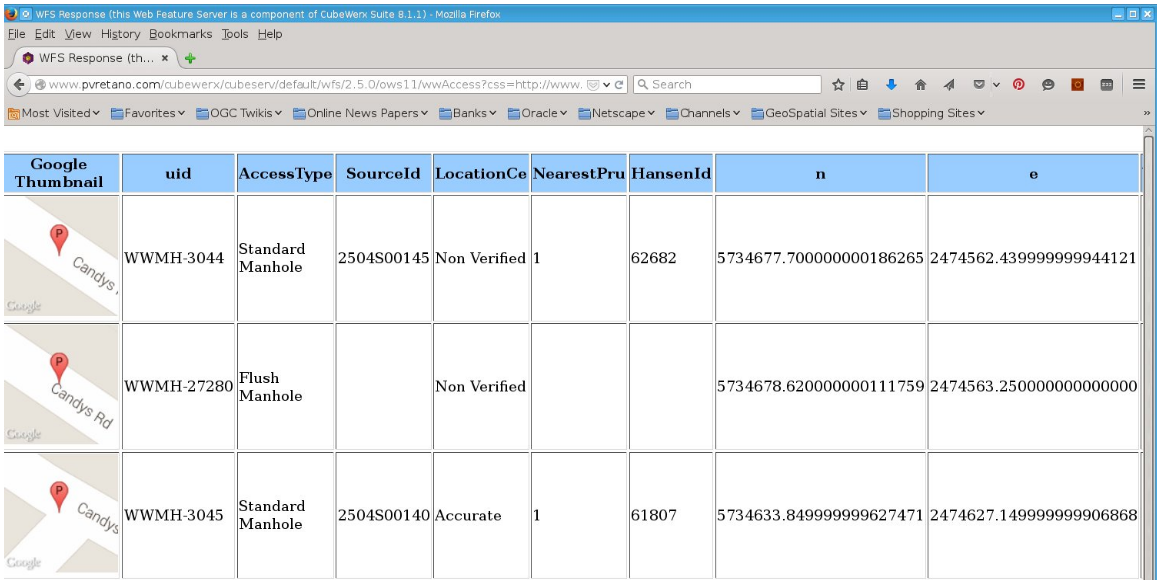
Figure 3 shows what the output of this request looks like in HTML.

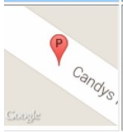
In order to force XML output in the Firefox browser we include the “outputFormat” or “f” query parameters as well:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess?bbox=-43.4380,172.6489,-43.3947,172.7108&f=application/gml%2Bxml;%20version=3.2

For the XML output, notice that each feature in the response is accompanied by a number of hypermedia controls.

Figure 3 – REST WFS query response in HTML


WFS Response (this Web Feature Server is a component of CubeWerx Suite 8.1.1) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
WFS Response (th...
www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess?css=http://www.
Search
Most Visited
Favorites
OGC Twikis
Online News Papers
Banks
Oracle
Netscape
Channels
GeoSpatial Sites
Shopping Sites

Google Thumbnail	uid	AccessType	SourceId	LocationCe	NearestPru	HansenId	n	e
	WWMH-3044	Standard Manhole	2504S00145	Non Verified	1	62682	5734677.700000000186265	2474562.439999999944121
	WWMH-27280	Flush Manhole		Non Verified			5734678.620000000111759	2474563.250000000000000
	WWMH-3045	Standard Manhole	2504S00140	Accurate	1	61807	5734633.849999999627471	2474627.149999999906868

Following any hypermedia control where *rel=”service”* will retrieve the capabilities document of the service that offers this feature.

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11

Following any hypermedia control where rel=”collection” will access the feature type or collection that contains this feature and execute a default query.

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess

Following any control where rel=”alternate” will retrieve an alternative representation of the same feature. For example:

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.34709.BA89DFD04C0B73161F20020000?outputFormat=application/vnd.geo%2Bjson

will retrieve the feature in GeoJSON while

Copyright © 2016 Open Geospatial Consortium.

http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.34709.BA89DFD04C0B73161F20020000?outputFormat=application/x-bxfs%2Bxml;%20version=%220.0.3%22

will retrieve the feature in BXFS (see OGC 07-004).

6.7 Transactions

Like feature access (see 7.6) the WFS REST binding segregates feature management or transactions into simple and complex classes.

Simple feature management is the manipulation of single features using HTTP's POST, PUT and DELETE methods as one would expect in a service using the REST architectural style (see 6.1.1, 6.2).

Complex feature management involves changes that operate on multiple features not necessarily all of the same type using either batch or transaction semantics. The WFS offers two approaches for performing complex transactions. The first is to simply use the methods currently defined – that is posting an XML document containing a wfs:Transaction element to the server. The second approach uses a transaction factory to create a transaction resource that can then be manipulated use the standard CRUD methods.

In the UCR thread of OGC Testbed 11, transactions were used in the enterprise synchronization scenario to update features on coordinating nodes. Primarily simple transactions were used. Complex feature management the REST interface was not tested in the testbed. For details about the enterprise synchronization scenario please see OGC 15-010 Testbed 11 Summary Report of Findings for WFS-T Information Exchange Architecture ER.

7 Accomplishments

In this Engineering Report, we tried to summarize best practices for the use of REST in existing and proposed OGC Web services standards. The most significant results from OGC Testbed 11 involved implementations of WFS 2.5, a valuable OGC standard and also a possible guide to incorporating REST principles and practices (as well as use of JSON representations) in other OGC Web services. In the absence of clear guidance and compelling motivation for standardized RESTful geo-services, it is clear that every working group will go its own way to develop API's, depending on their requirements, availability of development resources or knowledge of the subject. This may preserve specific characteristics of OGC service types, but will tend to lose the value of uniform interfaces and much of the ease of implementation that REST can bring. Clearly, the development of a uniform RESTful API binding for all OGC services will require both a commitment to standardization and some re-thinking of each OGC service to re-cast capabilities in a new light. The work should not so much be a conversion of existing services as a new understanding of how OGC service capabilities can be made available.

8 Future work

The only way to validate such guidelines is to apply them in a sandbox project. The subject is so vast that a entire interoperability project could be set up for this topic.

Some ingredients of a broadly applicable OGC REST binding, such useful linking, versioning, or asynchronous behavior still need to be analyzed and refined.

Copyright © 2016 Open Geospatial Consortium.

Annex A: List of OGC documents dealing with REST

Date	Title	OGC #	Class	Status
08-2007	RESTful OGC Services		Presentation	
08-2007	REST Analysis SC		Presentation	
09-2007	RESTifying The WCS		Presentation	Draft
09-2007	RESTful OGC Services		Presentation	
03-2008	RESTFul Workflows		Presentation	
12-2008	RESTful WCS		Presentation	Draft
12-2008	Rest and RPC		Presentation	Draft
12-2008	WMTS. KVP, SOAP and RESTful. TC Valencia		Presentation	
12-2008	RESTful geospatial services		Presentation	Draft
12-2008	WMTS. KVP, SOAP and RESTful. TC Valencia		Presentation	
04-2009	WMTS Practical and RESTful		Presentation	
04-2009	RESTful WPS		Presentation	
04-2009	OWS-6 RESTful Security ER		Presentation	
04-2009	OWS-6 DSS Engineering Report – SOAP/XML and REST in WMTS	09-006	Eng. Spec	Public Engineering Report
04-2009	09-006 OWS-6 DSS Engineering Report – SOAP/XML and REST in WMTS		Presentation	
04-2009	RESTful Security Demo Slides		Presentation	
04-2009	RESTful Demo – Building at Louis Armstrong Airport – New Orleans		Presentation	
05-2009	OWS-6_DSS_EngineeringReport-SOAP/XML_and_REST_in_WMTS		Presentation	
05-2009	OWS-6RESTful Security Presentation		Presentation	
05-2009	OWS-6 RESTful Security Demo Video (pdf)		Presentation	
05-2009	OWS-6 RESTful Security Demo Video (m4v)		Presentation	
05-2009	OWS-6 RESTful Scenario Video (pdf)		Presentation	
05-2009	OWS-6 RESTful Scenario Presentation (pdf)		Presentation	
05-2009	OWS-6 RESTful Workflows (m4v)		Presentation	
05-2009	OWS-6 RESTful Security Demo Video		Presentation	
05-2009	OWS-6 RESTful Security Requirements		Presentation	
05-2009	OWS-6 RESTful Security Presentation		Presentation	
05-2009	OWS-6 RESTful Scenario Presentation		Presentation	
05-2009	OWS-6 RESTful Workflow and Security (Vightel inputs for Ers)		Presentation	
05-2009	OWS-6 RESTful Workflow Architecture ER		Presentation	
12-2009	RESTFul Workflows, Next		Presentation	
12-2009	RESTful Synoptic View		Presentation	
12-2009	OWS-6 RESTful Workflows Movie		Presentation	
05-2011	Geoservices REST API Candidate Standard	11-049	Eng. Spec	Pending

06-2011	Geoservices REST Specification		Presentation	
06-2011	Geoservices REST API		Presentation	
07-2011	A REST binding for WFS 2.0	11-080	Change Request	Pending
07-2011	Comments on the ESRI GeoServices REST Specification Version 1.0 as the basis for OGC REST GeoServices		Presentation	
09-2011	12-062r2 GeoServices REST API – relationship with OGC baseline		Presentation	
09-2011	Geoservices REST SWG closing plenary report		Presentation	
09-2011	REST SC		Presentation	
09-2011	REST Common Patterns		Presentation	
09-2011	REST SC Report to TC		Presentation	
09-2011	Proposed Charter for RESTful Policy SWG		Presentation	
10-2011	Geospatial REST API SWG		Presentation	
Directory			Presentation	N/A
11-2011	RESTful Services Policy SWG		Legal Documents	Approved
11-2011	Geoservices REST SWG agenda and slides		Presentation	
01-2012	Geospatial REST API SWG		Presentation	
02-2012	RESTful Policy SWG		Presentation	
03-2012	Austin TC Meeting – RESTful Policy SWG SWG		Presentation	
03-2012	RESTful Policy SWG		Presentation	
05-2012	Geospatial REST API SWG		Presentation	
09-2012	RESTful WFS Presentation		Presentation	
09-2012	OGC standards in various domains of interest		Presentation	Draft
11-2012	Geoservices REST API SWG		Presentation	
12-2012	WCS Extension – REST Protocol Binding		Eng. Spec	
01-2013	WCS Extension – REST Protocol Binding		Presentation	
01-2013	GeoServices REST SWG Update		Presentation	
01-2013	20121221 WCS Rest Protocol Extension (draft)		Eng. Spec	RFC Proposal
03-2013	GeoServices REST API – Part 1: Core	12-054r2	Eng. Spec	Pending
03-2013	GeoServices REST API – Part 2: Catalog	12-055r2	Eng. Spec	Pending
03-2013	GeoServices REST API – Part 3: Map Service	12-056r2	Eng. Spec	Pending
03-2013	GeoServices REST API – Part 4: Feature Service	12-057r2	Eng. Spec	Pending
03-2013	GeoServices REST API – Part 5: Geometry Service	12-058r2	Eng. Spec	Pending
03-2013	GeoServices REST API – Part 6: Image Service	12-059r2	Eng. Spec	Pending
03-2013	GeoServices REST API – Part 7: Geoprocessing Service	12-060r2	Eng. Spec	Pending
03-2013	GeoServices REST API – Part 8: Geocoding Service	12-061r2	Eng. Spec	Pending
03-2013	GeoServices REST API – JSON Schemas and Examples	12-068r2	Technical Information	Pending
03-2013	GeoServices REST API – Relationship with the OGC standards baseline	12-062r2	Technical Information	Pending
03-2013	GeoServices REST API – RFC Comments	12-164	Technical Information	Pending
04-2013	GeoServices REST API – SWG response to justification comments for No-Votes	13-031	Technical Information	Pending

Copyright © 2016 Open Geospatial Consortium.

05-2013	GeoServices REST API – SWG response to justification comments for No-Votes	13-031r1	Technical Information	Pending
05-2013	12-164 GeoServices REST API – RFC Comments		Technical Information	
09-2013	RESTful Encoding of Ordering Services & Download protocol For EO Products		Presentation	
02-2014	RESTful Encoding of Sensor Planning Service for Earth Observation Satellite Tasking	14-012	Eng. Spec	Pending
03-2014	RESTful Encoding of Sensor Planning Service for Earth Observation Satellite Tasking		Presentation	
03-2014	RESTful Encoding of Sensor Planning Service for Earth Observation Satellite Tasking		Presentation	
04-2014	13-042_OGC_RESTful_Encoding_of_Ordering_ Services_Framework_For_Earth_Observation_Products as BP		Eng. Spec	Best Practices
04-2014	OGC RESTful Encoding of Ordering Services Framework For Earth Observation Products	13-042	Eng. Spec	Best Practices
07-2014	14-012r1 OGC RESTful encoding of OGC Sensor Planning Service for Earth Observation satellite Tasking		Eng. Spec	Best Practices
07-2014	RESTful Encoding of Sensor Planning Service for Earth Observation Satellite Tasking	14-012r1	Eng. Spec	Best Practices
03-2015	REST binding for WFS 2.0		Eng. Spec	
05-2015	WaterML2.0 part 2 – RESTful API and JSON encoding	15-033	Eng. Spec	Pending
06-2015	TB-11 REST in OGC Services ER		Presentation	
06-2015	WaterML2.0 part 2 – RESTful API and JSON encoding Best Practice		Presentation	

Annex B: revision history

Date	Release	Editor	Primary clauses modified	Description
2015-06-10	0.1	Frédéric Houbie	all	first version of complete ER
2015-06-14	0.2	Joan Masó	WMTS subclause	WMTS considerations included
2015-06-30	0.3	Peter Vretanos	WFS 2.5	Description of WFS 2.5 REST
2015-08-21	0.9	Frédéric Houbie	all	General editing
2015-09-13	1.0	Josh Lieberman, Lew Leinenweber	all	General editing and contrast of REST and OGC service styles
2016-01-06	1.1	Frédéric Houbie	all	Move list of REST documents at OGC as an annex, General editing
2016-01-06	1.1	Scott Simmons	All	Finalize for publication

Copyright © 2016 Open Geospatial Consortium.

Bibliography

[1] http://www.restapitutorial.com/ under Creative Commons Attribution-ShareAlike 4.0 International License

[2] Principles of good RESTful API Design (source : http://codeplanet.io/principles-good-restful-api-design/)

[3] Consumer Centric REST API : https://thomashunter.name/consumer-centric-api-design/Consumer-Centric%20API%20Design%20v0.3.1.pdf