Testbed-12 REST User Guide

Table of Contents

Publication Date: 2016-mm-dd

Approval Date: 2016-mm-dd

Posted Date: 2016-11-14

Reference number of this document: OGC 16-057

Reference URL for this document: http://www.opengis.net/doc/PER/t12-A060

Category: User Guide

Editor: Jeff Harrison

Title: Testbed-12 REST Users Guide

COPYRIGHT

IMPORTANT	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.
NOTE	This document is a user guide created as a deliverable in an OGC Interoperability Initiative as a user guide to the work of that initiative and is not an official position of the OGC membership. There may be additional valid approaches beyond what is described in this user guide.

POINTS OF CONTACT

Name	Organization
Jeff Harrison (Editor)	The Carbon Project
Simon Jirka	52°North GmbH
Christoph Stasch	52°North GmbH
Peter Vretanos	CubeWerx
Keith Pomakis	CubeWerx
Craig Bruce	CubeWerx
Charles Heazel	Wisc Enterprises
Stephane Fellah	Image Matters
Dave Wesloh	NGA
George Percivall	OGC
Ingo Simonis	OGC
Joan Maso	CREAF
Peter Baumann	rasdaman GmbH
Andreas Matheus	Secure Dimensions

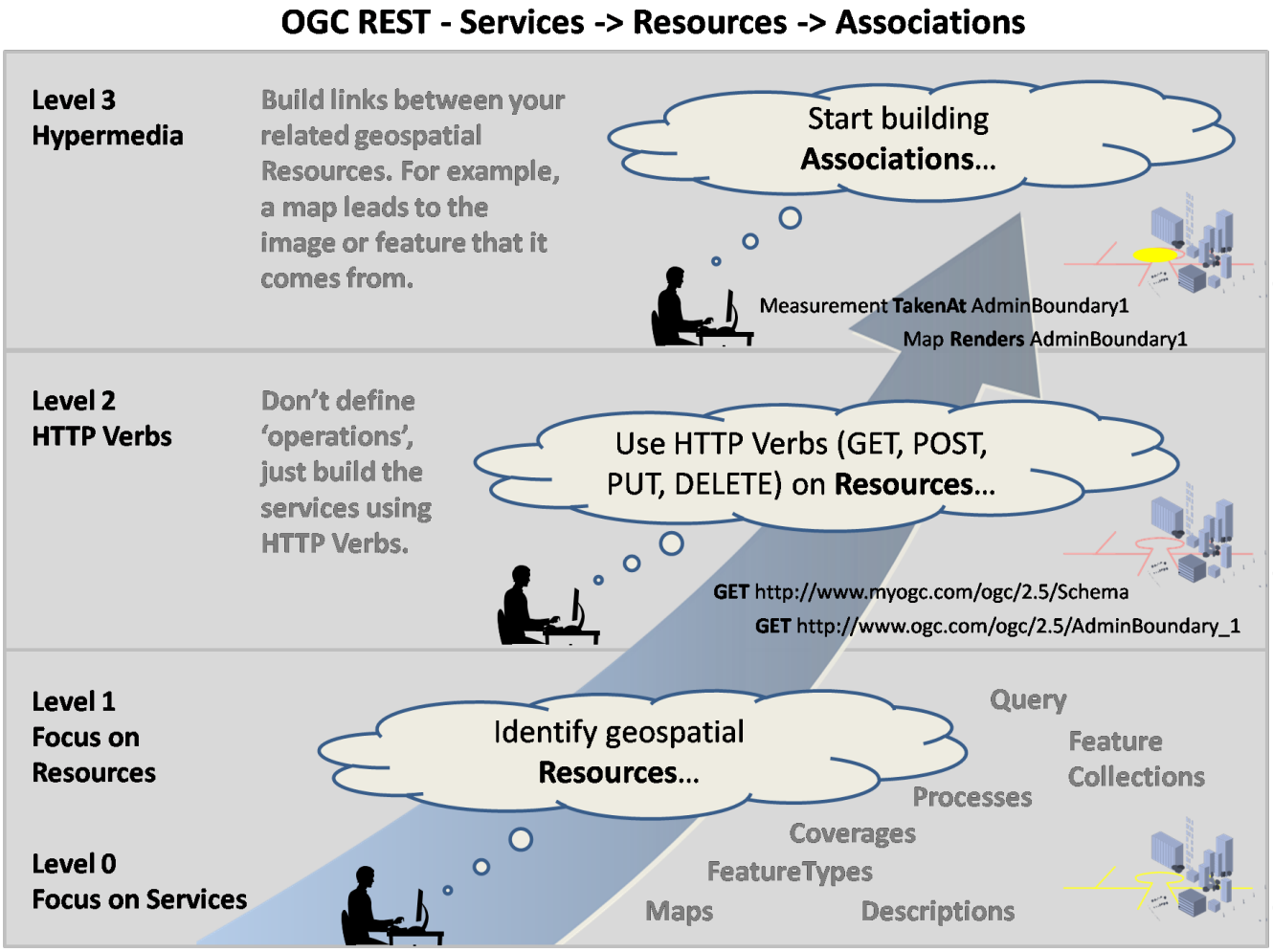
Chapter 1. Executive Summary

This document presents a User's Guide for open geospatial REST. It complements Testbed 12 architecture documents to assist enterprise implementers. The content is presented from the perspective of a person or organization trying to decide to use (or not use) open geospatial REST to offer data and services.

There are almost as many definitions of REST as people using it. Simply put, REST is a method of discovering, accessing and updating geospatial content using the standards of the World Wide Web – Resources, HTTP methods (GET, PUT, DELETE etc.) and hypermedia controls. However, there's often confusion in the geospatial community over 'What is RESTful?'. Lengthy email threads have discussed whether you must use hypermedia, or structured URLs. Previous efforts have even gone to great length to list diverse OGC documents with 'REST' in the title. These discussions don't help the central challenge – how do we make it easy to access open geospatial information, and answer key questions about the world around us?

To meet this challenge, it's important to recognize that REST isn't a single, monolithic approach. It doesn't even have to apply only to the Web. However, for the purposes of this guide we limit REST to the Web (i.e. HTTP) - and then break it down into four Levels. These Levels are derived from the Richardson Maturity Model (RMM).

The four Levels progress from Services to a focus on geospatial Resources, HTTP Verbs and linking Resources. A simplified representation of this Services → Resources → Associations progression is presented in the graphic below. Basic guidance is also provided at each Level. It's important to recognize that these Levels aren't discrete and exclusive – they build upon each other to help geospatial interoperability.



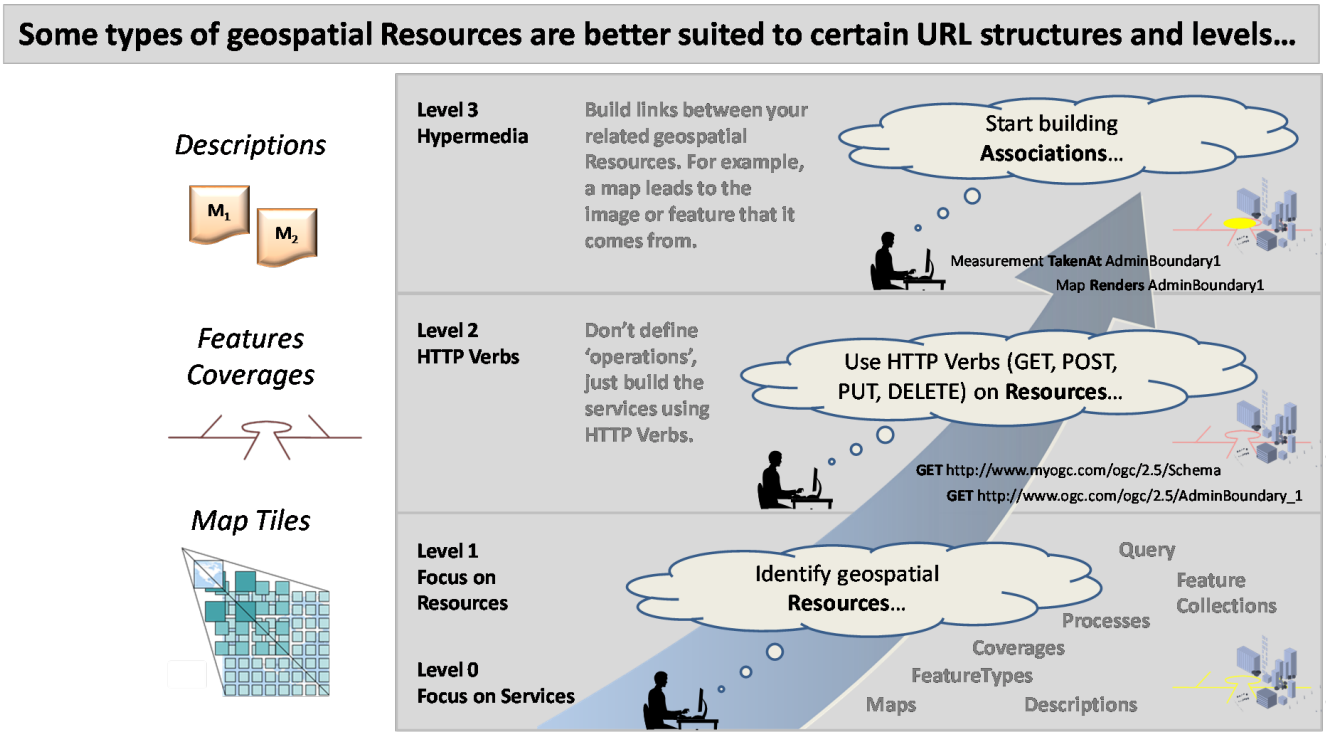
Starting at Level 0 the focus is on making requests to 'Services' using HTTP as a transport system for our remote interactions, often with XML. Level 0 represents the current state of many OGC Web Services. At Level 1 an enterprise can start to identify open geospatial Resources and then make requests to the Resources. An open geospatial Resource is geospatial information, like a feature or satellite imagery data, that can be identified by a URL.

Since REST is Resource oriented and the standard OGC web architecture is service oriented some adaptation of the existing OGC service model is required. However, rather than completely redefining OGC Web Services, Testbed 12 has taken an evolutionary approach in developing the REST bindings that reuse as much of OGC Web Services functions as possible – but update things to work RESTfully.

At Level 2, a geospatial enterprise may begin using the HTTP 'verbs' the same way they are used in HTTP itself. This means we don't define unique 'Operations' like INSERT, UPDATE, DELETE to access or modify Features and other open geospatial Resources. Instead, we use HTTP methods like GET, POST, PUT, DELETE. Using these HTTP methods can make many things easier for API developers and client applications.

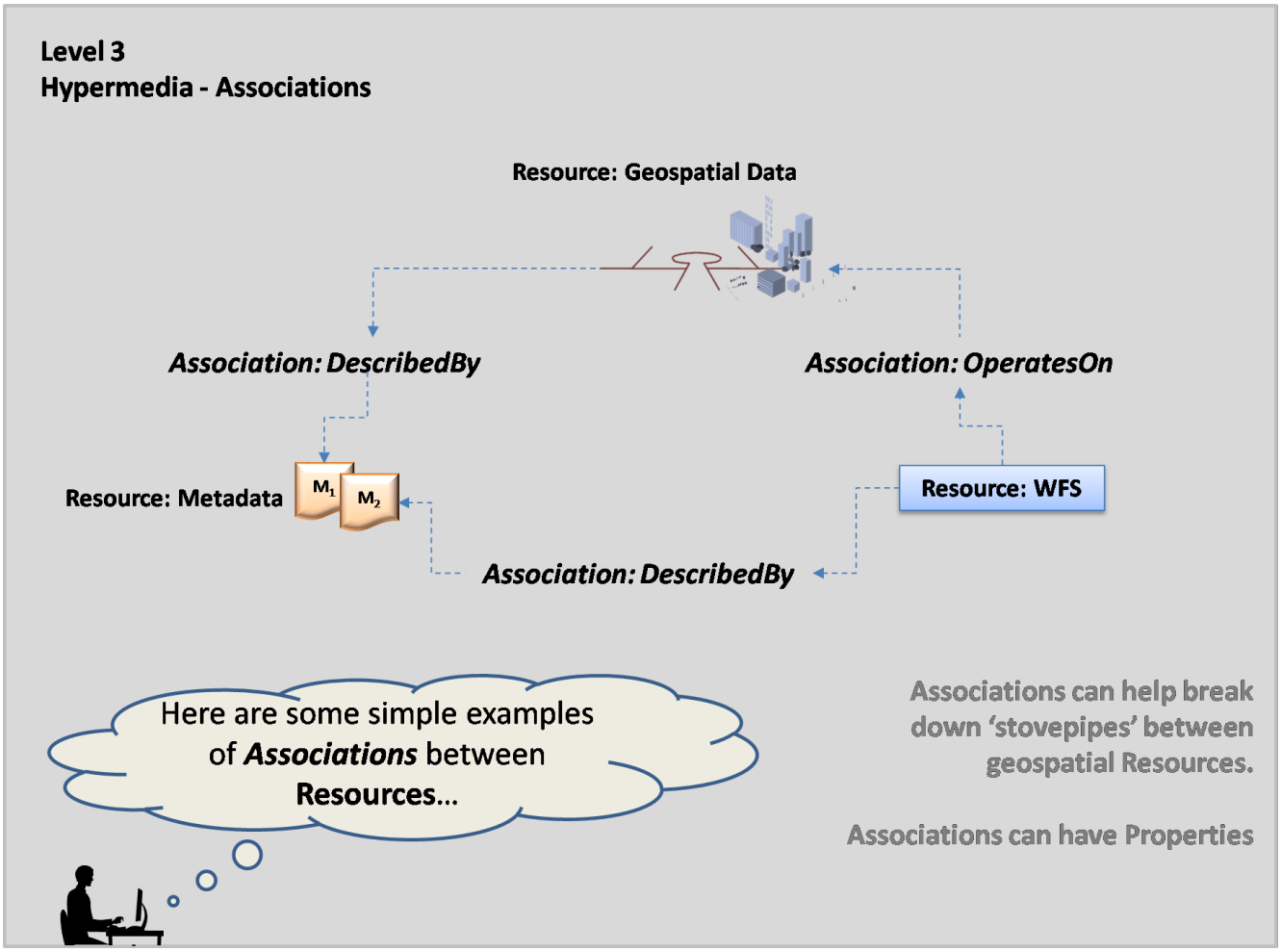
With Resources defined and HTTP verbs in use, at Level 3 we begin to focus on hypermedia and Associations. That means we start building links between related open geospatial Resources. For example, hypermedia links may be used to link metadata to different information models or schemas. In addition, a Map Resource may have Associations to the satellite image data that it comes from.

It's also very important to recognize that open geospatial Resources can include very complex information about the Earth – such as electro-optical, infrared, multispectral, hyperspectral and radar satellite measurements, map tiles, sensor observations and much more. This means that as much as we want all Resources to fit neatly into Level 1, 2 or 3, the reality is some open geospatial Resources are better suited to certain URL structures or Levels.



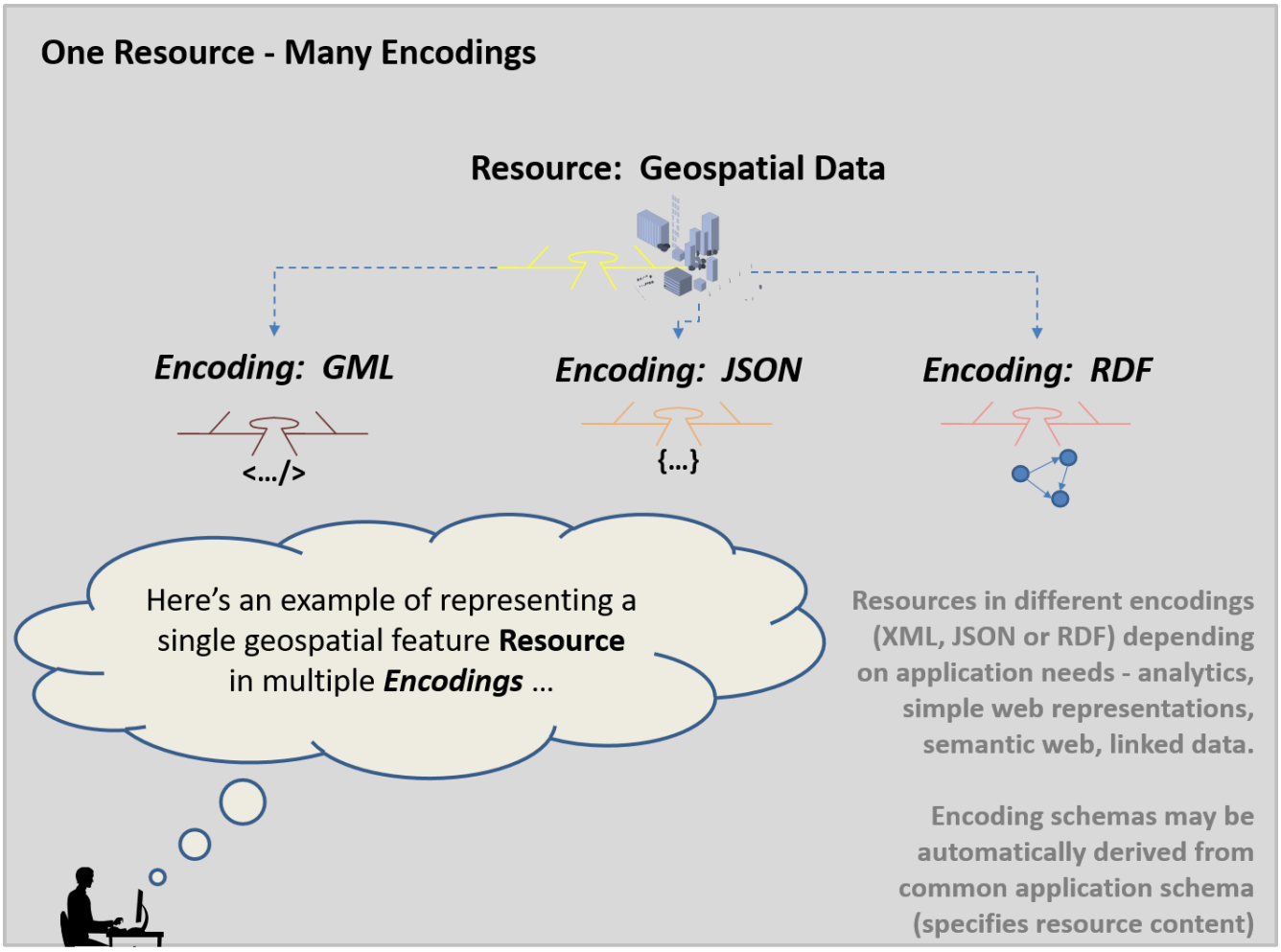
As we consider URL structures another key concept for open geospatial Resources is the URI Template. A URI template establishes the structure of a URI (Resource URL) by indicating what values (parameters) can be added and what they represent. URI templates are useful at all Levels, depending on the open geospatial Resource type and access paths needed.

Finally, at Level 3 enterprises can start to build Associations between open geospatial Resources. As of Testbed 12, Associations are just beginning to be defined in open geospatial REST but they are essentially linkages between Resources. These linkages can help break down the 'stovepipes' between open geospatial Resources. Some examples are included in the graphic below.

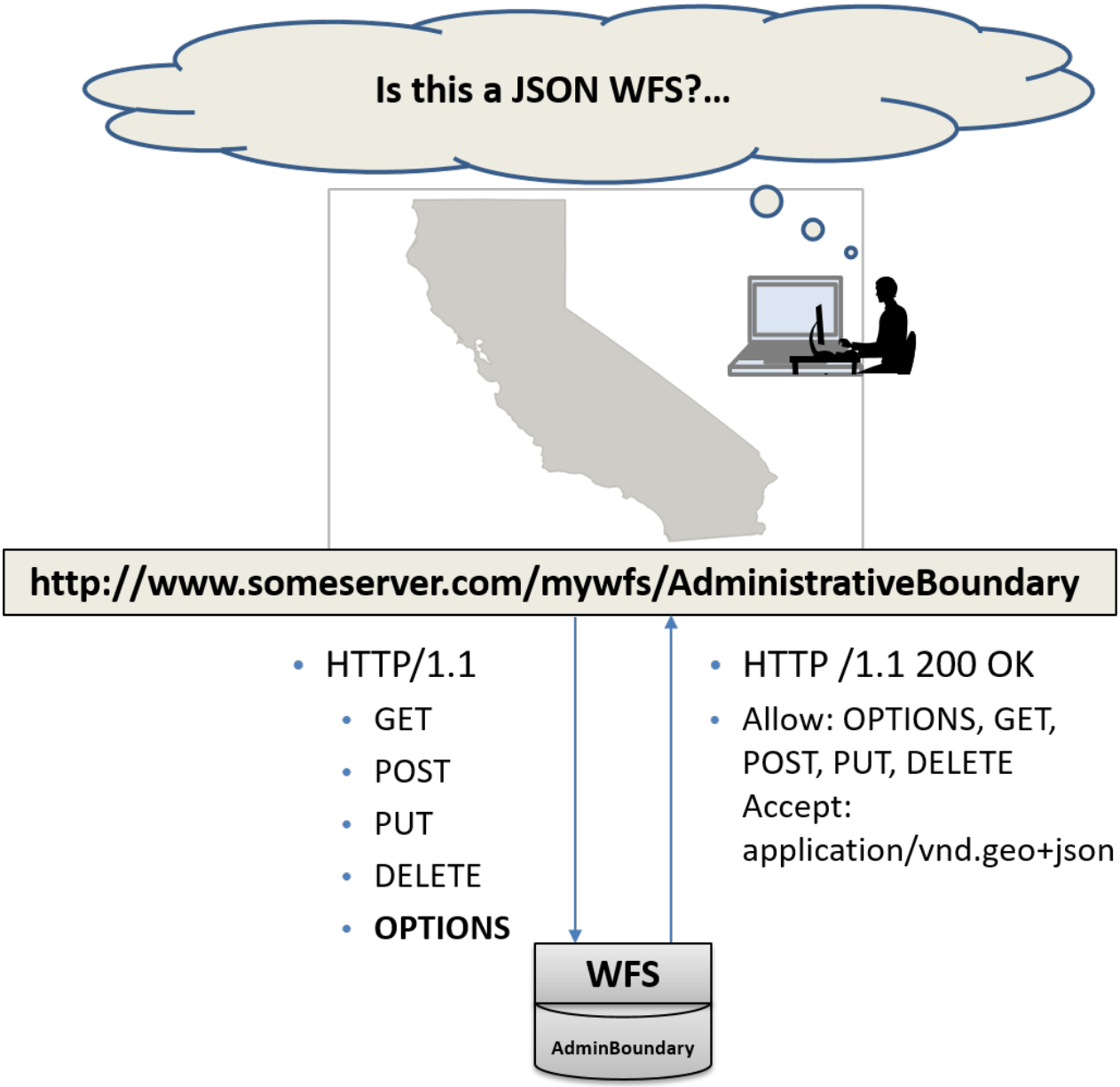


As geospatial enterprises begin to implement the four Levels discussed it is possible to make it easier to access open geospatial information, and answer key questions about the world around us.

Another important finding of Testbed 12 is that open geospatial REST allows applications to select the output format most useful to their mission. For example, Resources can be represented in different encodings such as XML, RDF or JSON, depending on whether applications need complex analytics, linked data or simpler representations for web display. A simple examples is shown below.



Finally, Testbed 12 assessed that most of the pieces are in place for a REST WFS to support geospatial intelligence features as GeoJSON-only - without the need for a new profile document. Several issues must be addressed, however, this type of GeoJSON WFS would simplify application development and open the potential to begin moving from GML to linked features and observations in JSON.



In summary, the technology integration experiments conducted in Testbed 12 showed that REST was able to represent a variety of real world phenomena as open geospatial Resources. The Resources can then be accessed and updated using the language of the World Wide Web - greatly simplifying development and deployment for geospatial enterprises.

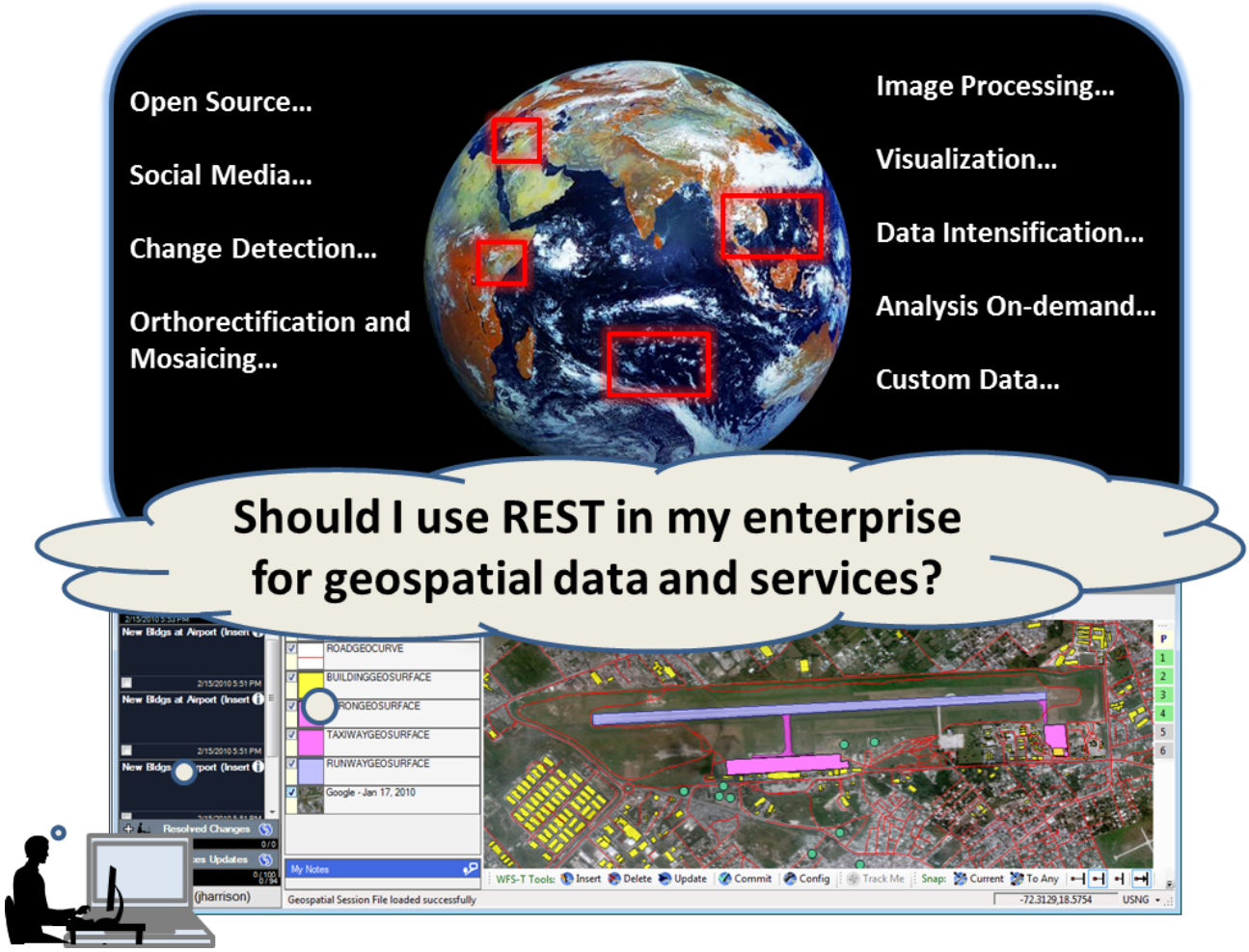
Chapter 2. Introduction

In 1999 OGC started developing a suite of web services for geospatial interoperability. These included web map services (WMS), web map tiling services (WMTS), web feature services (WFS), web coverage services (WCS), catalogue services (CSW), the Sensor Web services, etc.

For the most part these were developed using HTTP as a tunneling mechanism for remote interactions, usually sending Key Value Pairs (KVP) or plain old XML (POX) back and forth. As OGC web services started to use SOAP it was basically the same mechanism, the only difference was the messages were wrapped in an XML 'envelope'. All these methods worked, but more and more specialized tools were required to access and use open geospatial data – costing time and money.

As years passed simpler alternatives became more popular. In particular, an approach called 'REST' began to be explored throughout the geospatial community. REST uses the standard language of the World Wide Web, HTTP, to discover, access and update geospatial resources.

This document presents a User's Guide for open geospatial REST. The content is presented from the perspective of a person or organization trying to decide to use (or not use) open geospatial REST to offer data and services. It complements the Testbed 12 REST Architecture Engineering Report (16-035) that is designed to assist enterprise implementers.



This document is created as a deliverable in an OGC Interoperability Initiative, Testbed 12. It is anticipated this document will undergo OGC Technical Committee Standards and Domain Working Groups review and may be subject to regular updates. This document consists of the following

sections:

- **User Guidance**
  - REST and Open Geospatial Resources
  - Advantages and Principles
  - Key Terms
- **Implementer Guidance**
  - Use Nouns and Don’t Use Verbs
  - Use Well-Known Resource Classes
  - Use HTTP Status Codes
  - Use Well-Known URL Templates and Access Paths
  - JSON Examples
  - API Management

This document IS NOT a technical architecture or an OGC interface specification. The content is presented from the perspective of a non-technical reader.

Chapter 3. User Guidance

This section presents information from the perspective of a person trying to decide to offer geospatial information and services through REST as open geospatial Resources. This section also covers the principles, advantages and challenges of REST-based geospatial architecture.

3.1. REST and Open Geospatial Resources

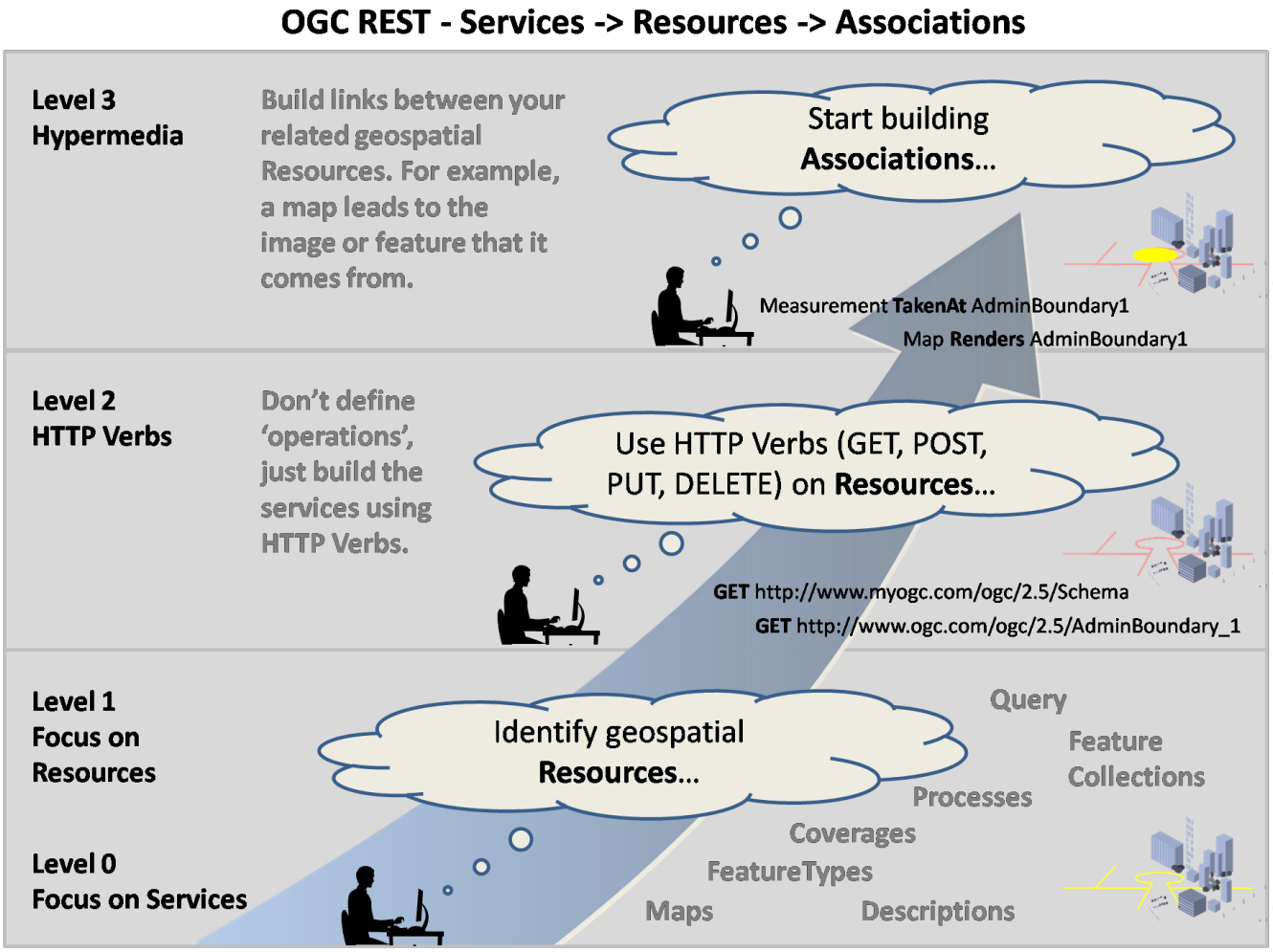
There are almost as many definitions of REST as people using it. As stated above, REST is a method of discovering, accessing and updating geospatial content using the standards of the World Wide Web – Resources, HTTP methods (GET, PUT, DELETE etc.) and hypermedia controls.

However, there's often confusion in the geospatial community over 'What is RESTful?'. Lengthy email threads have discussed whether you must use hypermedia, structured URLs or whether it is 'allowed' to include 'query parameters' in a URL. Previous efforts have even gone to great length to list diverse OGC documents with 'REST' in the title.

These discussions don't help the central challenge – how do we make it easy to access open geospatial information, and answer key questions about the world around us?

To meet this challenge, it's important to recognize that REST isn't a single, monolithic approach. It doesn't even have to apply only to the Web. However, for the purposes of this guide we limit REST to the Web (i.e. HTTP) - and then break it down into four Levels. These Levels are derived from the Richardson Maturity Model (RMM).

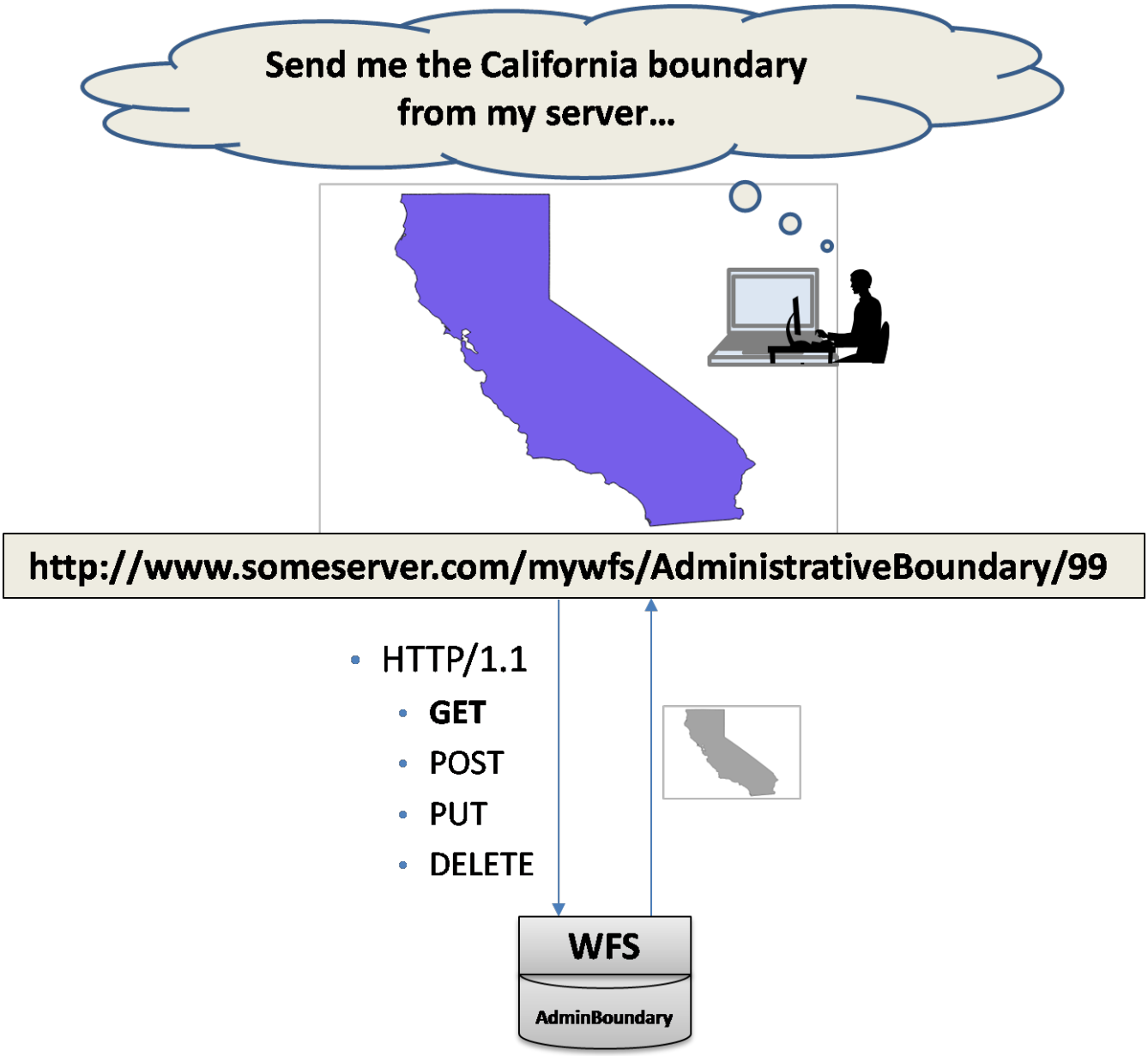
The four Levels progress from Services to a focus on geospatial Resources, HTTP Verbs and linking Resources. A simplified representation of this Services → Resources → Associations progression is presented in the graphic below. Basic guidance is also provided at each Level. It's important to recognize that these Levels aren't discrete and exclusive – they build upon each other to help geospatial interoperability.



Starting at Level 0 the focus is on making requests to 'Services' using HTTP as a transport system for our remote interactions, often with XML. Level 0 represents the current state of many OGC Web Services.

At Level 1 an enterprise can start to identify open geospatial Resources and then make requests to the Resources. An open geospatial Resource is geospatial information, like a feature or satellite imagery data, that can be identified by a URL.

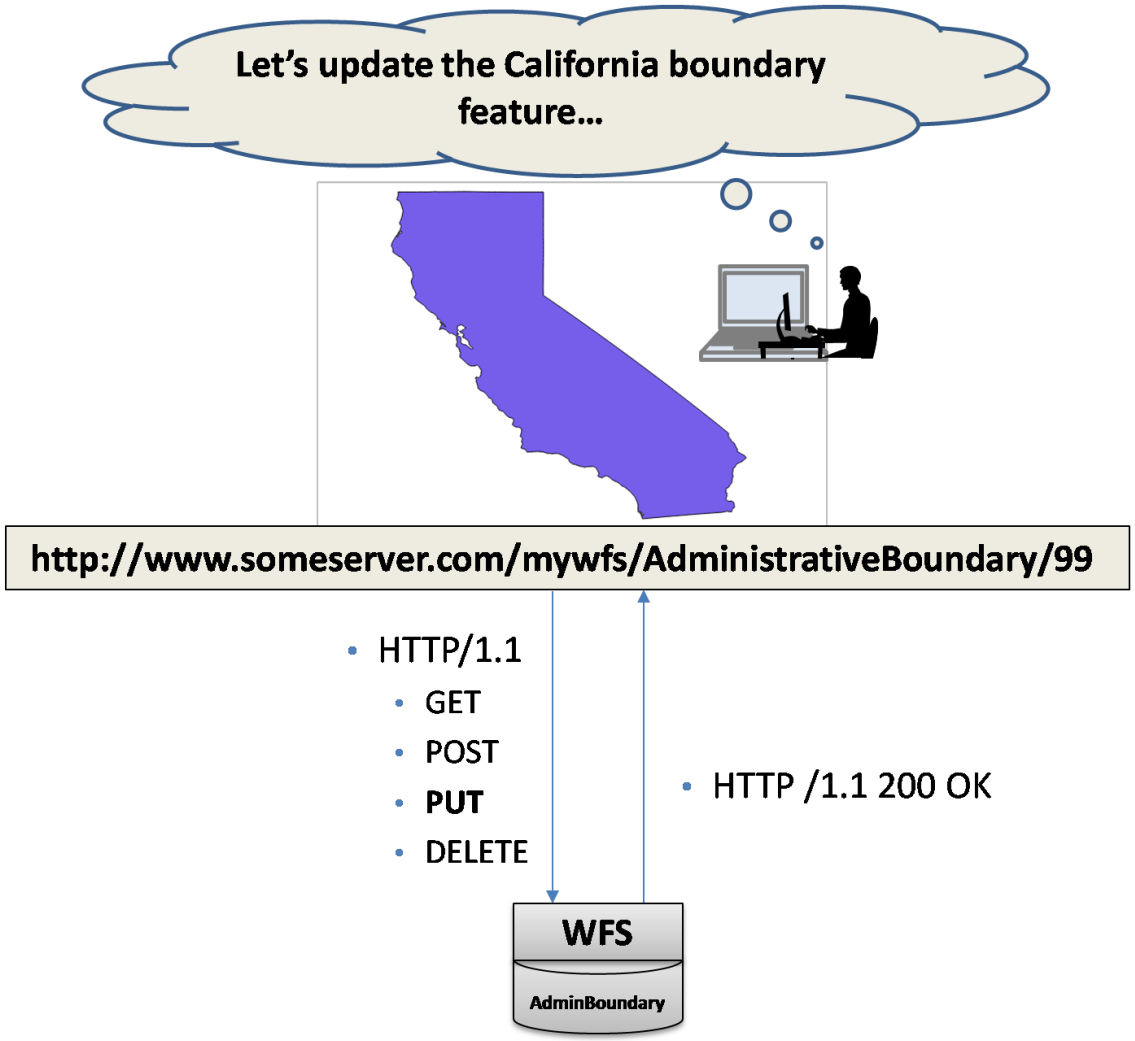
An example of making a request to a Feature Resource is shown below. In this example, an application needs to get the boundary of the state of California. To do this, it sends an HTTP request to the Resource at the URL indicated.



Since REST is Resource oriented and the standard OGC web architecture is service oriented some adaptation of prior OGC service model is required. However, rather than completely redefining OGC Web Services, Testbed 12 has taken an evolutionary approach in developing the REST bindings that reuse as much of OGC Web Services functions as possible – but update things to work RESTfully.

At Level 2, a geospatial enterprise may begin using the HTTP 'verbs' the same way they are used in HTTP itself. This means we don't define unique 'Operations' like INSERT, UPDATE, DELETE to access or modify Features and other open geospatial Resources. Instead, we use HTTP methods like GET, POST, PUT, DELETE.

Using HTTP verbs can make many things easier for API developers and client applications. For example, if an application needs to change something in the California boundary feature it simply sends an HTTP PUT with information to update the Feature Resource.



So, where possible, open geospatial REST uses HTTP methods for actions against Resources –

- GET
  - Retrieve a specific Resource from the server
  - Retrieve a collection of Resources from the server
- POST
  - Create a new Resource on the server
- PUT
  - Update an existing Resource on the server
- DELETE
  - Remove a Resource from the server
- OPTIONS
  - Used to get available options for Resources (what you can do with Resources)

HTTP Status Codes such as '200 OK', '400 Invalid Request', '404 Not Found' and all others are also

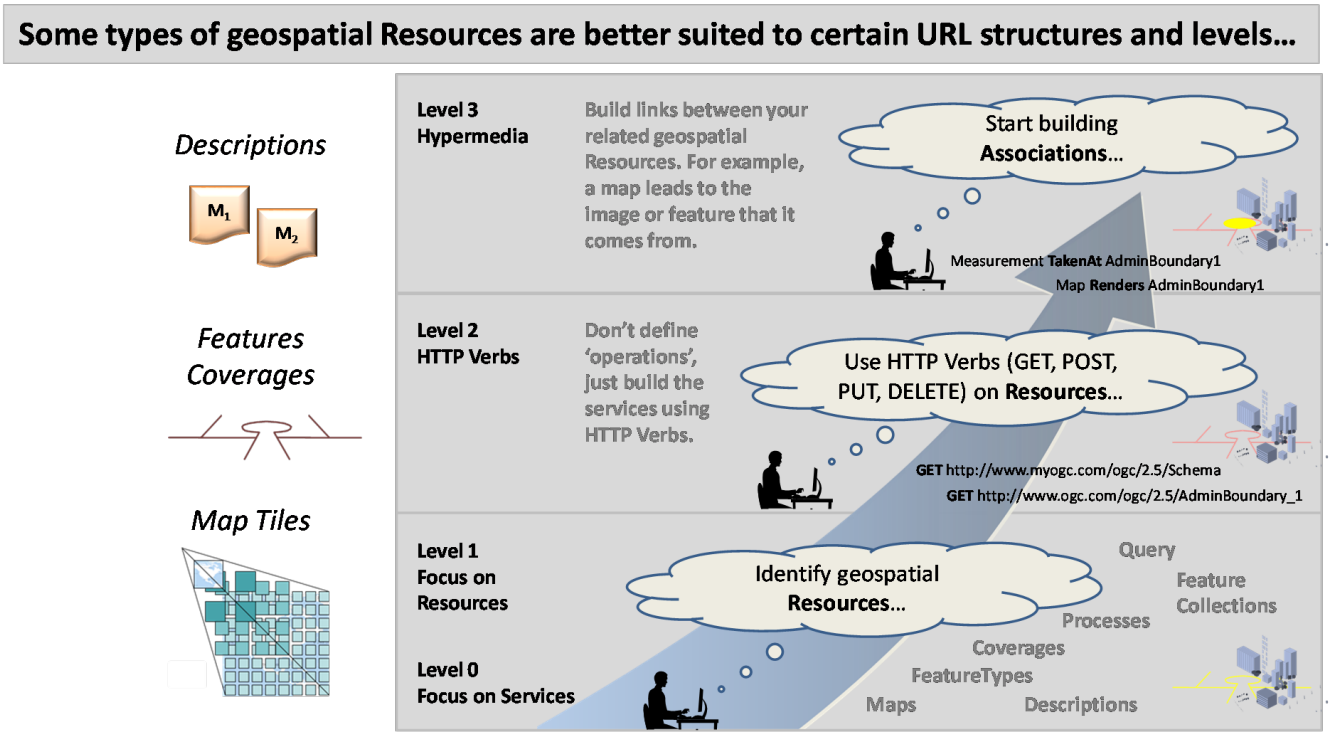
used in accordance with the HTTP 1.1 specification.

HTTP methods on resources are subject to authentication and access control (e.g. someone who does not have authorization to create new features would not see the POST methods when interrogating a resource with the OPTIONS method). Authentication and access control are envisioned to be layered on top of RESTful resources.

At this point, an enterprise architect may expect that all open geospatial Resources must available on a server. But this is not the case. Conceptually, and from the client's perspective, an open geospatial Resource is available even if serving it involves dynamic generation. For example, in the case of a REST WMTS it may be more practical to render Tile Resources on-the-fly than creating, storing and billions of small images. This is because to create and store all the possible Resources for a time-queryable REST WMTS you would have to multiply billions of tile positions by every allowed open-ended interval to fully pre-generate them. With 19+ digits of freedom on each end of an arbitrary time interval, you end up with ~10 possible REST Resource URLs, or quattuordecillions of them. So it may be more practical to generate some Resources 'on the fly'.

With Resources defined and HTTP verbs in use, at Level 3 we begin to focus is on hypermedia and Associations. That means we start building links between related open geospatial Resources. For example, hypermedia links may be used to link metadata to different information models or schemas. In addition, a Map Resource may have Associations to the satellite image data that it comes from.

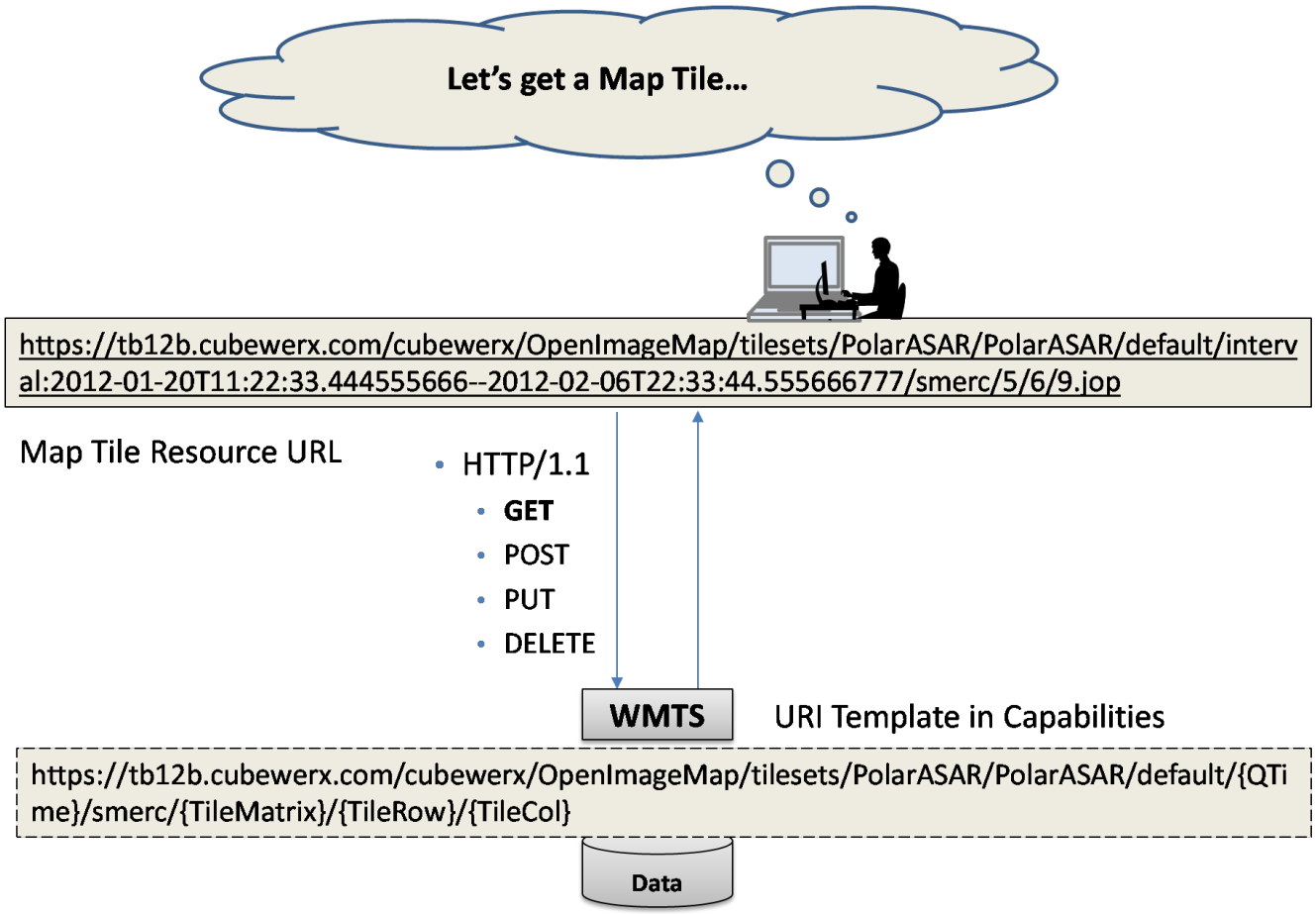
At this point it's very important to recognize that open geospatial Resources can include very complex information about the Earth – such as electro-optical, infrared, multispectral, hyperspectral and radar satellite measurements, map tiles, sensor observations and much more. This means that as much as we want all Resources to fit neatly into Level 1, 2 or 3, the reality is some open geospatial Resources are better suited to certain URL structures or Levels.



Using the example of a WMTS again, it wouldn't be practical to navigate to quattuordecillions of time-varying Tiles via hyperlinks at 'Level 3'. But a metadata description for open geospatial

Resources may contain many useful hypermedia controls to navigate to different Resources.

As we consider URL structures another key concept for open geospatial Resources is the URI Template. A URI template establishes the structure of a URI (Resource URL) by indicating what values (parameters) can be added and what they represent. The syntax of a URI template is to enclose the parameters in braces, like this {{example}}. Another example is provided below…



If we examine the URL in the example…

https://tb12b.cubewerx.com/cubewerx/OpenImageMap/tilesets/PolarASAR/PolarASAR/default/interval:2012-01-20T11:22:33.444555666—2012-02-06T22:33:44.555666777/smerc/5/6/9.jop

…we can see it is the result of a URI template that's otherwise fully opaque, which means clients shouldn't attempt to construct their own (beyond the prescribed variable substitutions). In this example the URI template reported by the Capabilities Resource (metadata) is:

https://tb12b.cubewerx.com/cubewerx/OpenImageMap/tilesets/PolarASAR/PolarASAR/default/{QTime}/smerc/{TileMatrix}/{TileRow}/{TileCol}

But the URI template could just as well have been:

https://foo.com/foo{TileCol}foo{QTime}foo{TileRow}foo{TileMatrix}

URI templates are useful at all Levels, depending on the open geospatial Resource type and access paths needed.

Since the client shouldn't attempt to interpret URI Templates and shouldn't attempt to construct

their own, whether or not the URL contains a version number or service identification and what its location and syntax is entirely up to the needs of the server. That's one of the powers of the URI Template - the client requires no knowledge of the structures of the URLs beyond the URI Template variables, and the server is free to use whatever URLs are most convenient for it.

Given this possible variability, a starting point is useful to answer three questions about relevant open geospatial Resources - "What are they?", "How do I get to them?", and "How do I get the things they may provide?".

To help answer these questions the Capabilities Resource provides a complete metadata document for other Resources offered. Interactions with a set of OGC Resources usually commences by accessing the Capabilities document.

Current OGC REST documentation describes two ways, the 'Base URL' and 'Service Root' to get to the Capabilities Resource. A Service Root is an actual URL where the Capabilities Resource exists. A Base URL is a URL prefix that the client appends parameters to in order to arrive at a valid request URL, including a request URL for the Capabilities Resource. This situation is the result of supporting previous methods while being new methods into the interoperability architecture.

The versions supported are also stated in the Capabilities Resource. If a service supports several versions, these may be advertised in different Capabilities documents provided at different endpoints, For example, for Resources associated with processing functions this would be -

http://my.url.com/wps-rest/1.0

and

http://my.url.com/wps-rest/2.0

For Resources associated with features this would be -

http://www.someserver.com/wfs/1.0.0

and

http://www.someserver.com/wfs/2.0

and so on.

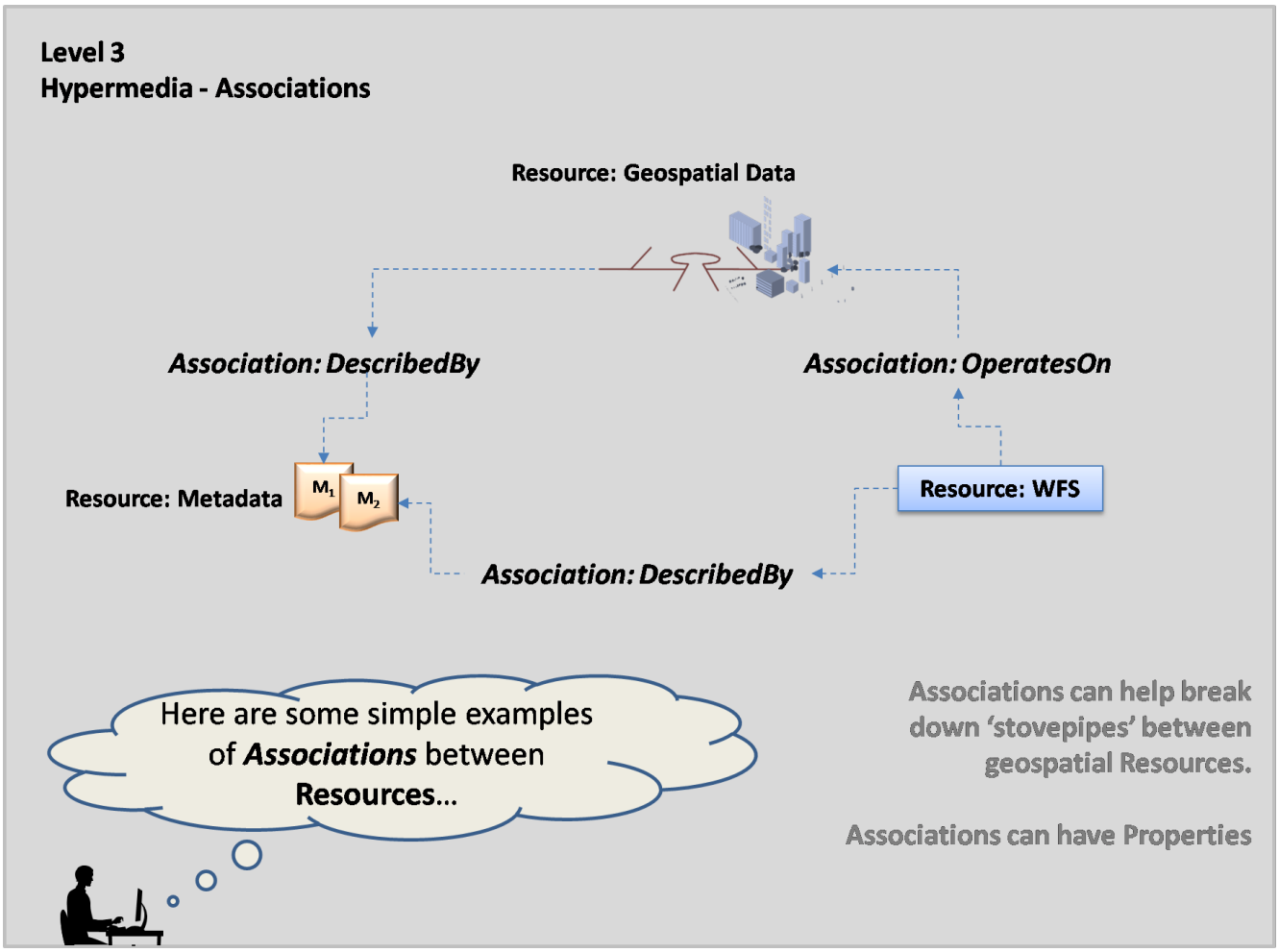
The Service Root and well-formed Base URLs bring up another key point. OGC Web Services have long-been criticized because nothing was available at the service URL - unless you appended a Request=GetCapabilities KVP. The starting points Service Root and well-formed Base URLs address this issue by providing both human and machine readable information about the Resources offered.

The Capabilities Resource should include hypermedia controls (i.e. links) that let a client application move from the Capabilities to the other Resources offered. Applications can also go directly to the Resources if they know how.

At Level 3 we can also begin to look at Resource constructs such as hypermedia controls in general. Hypermedia controls allow an application to obtain the URIs for Resources it needs by following

links in the representation of the Resources themselves - not by relying on out-of-band information like URI patterns given in documentation. For example, the response to a query for features should include links that tell the client how to get the capabilities document of the service that offers the feature; of which feature collection the feature is a member; which feature is next in the current results set; what alternative representations of the feature are available.

In addition, at Level 3 enterprises can start to build Associations between open geospatial Resources. As of Testbed 12, Associations are just beginning to be defined in open geospatial REST, but some examples are included in the graphic below.

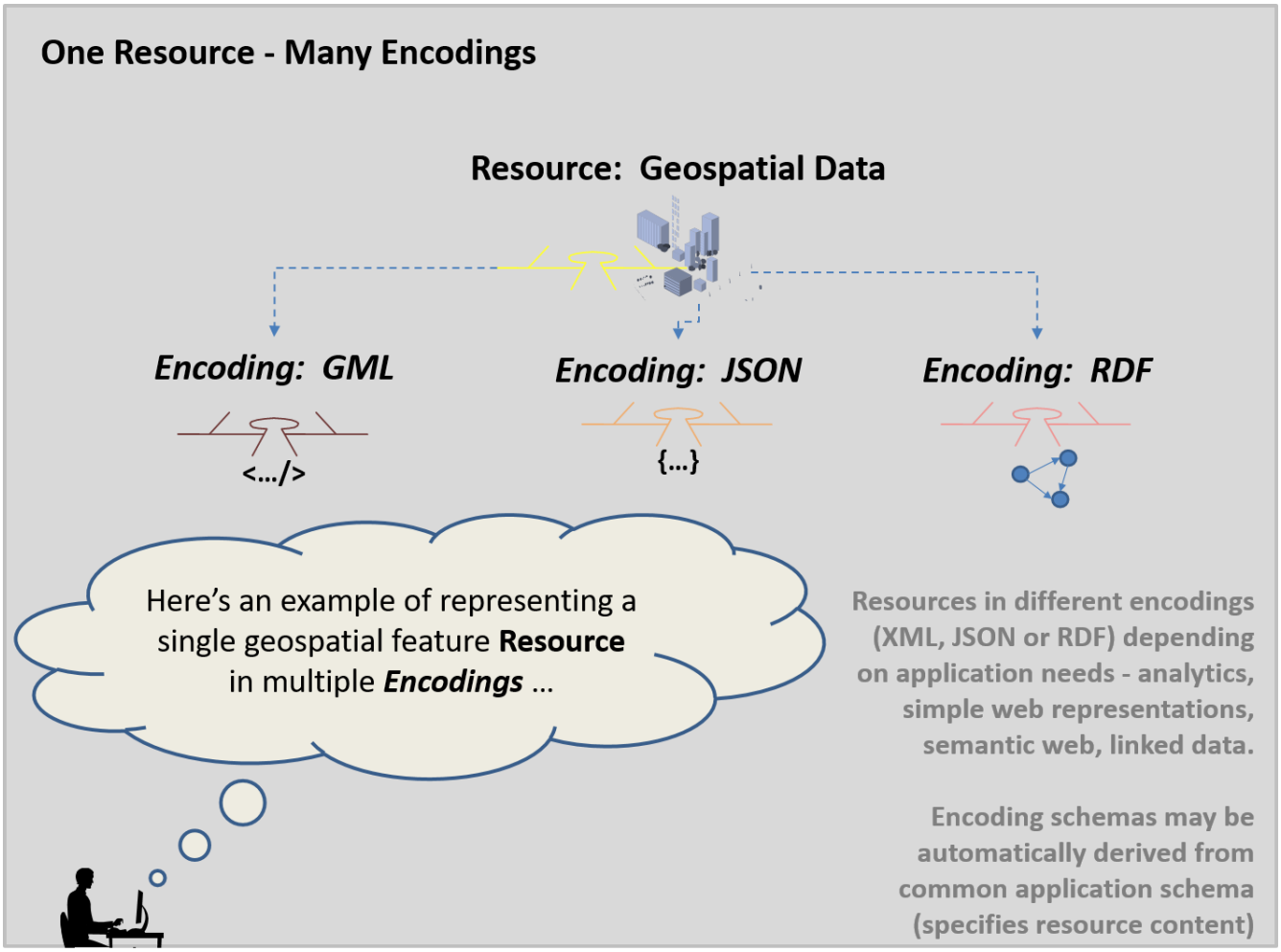


Associations are linkages between Resources, and can help break down the 'stovepipes' between open geospatial Resources. For example, an Association can indicate the source data set for a web map (the Association below is 'renders') -

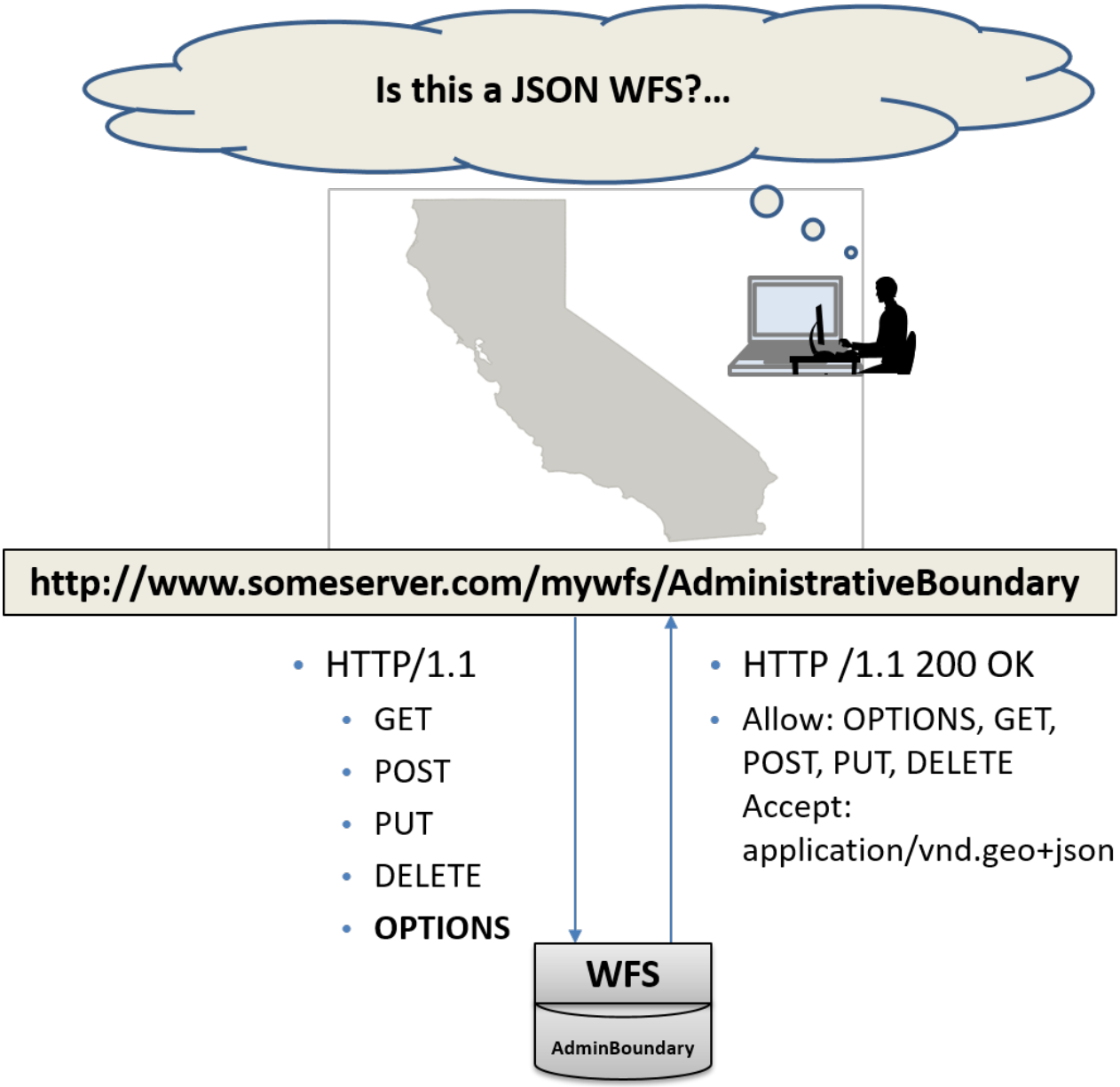
```
<Association rel="http://www.opengis.net/def/associationType/ogc/0/renders">
<Source type="http://www.opengis.net/def/objectType/ogc/0/wms/layer"
xlink:href="https://tb12.cubewerx.com/a011/cubeserv">
<Identifier>USGS.Elev_Contour</Identifier>
</Source>
<Target type="http://www.opengis.net/def/objectType/ogc/0/wfs/featureType"
xlink:href="https://tb12.cubewerx.com/a011/cubeserv?DATASTORE=USGS">
<Identifier
namespace="http://schemas.cubewerx.com/namespaces/null">Elev_Contour</Identifier
```

It should be noted that REST is often viewed as synonomous with JSON. But REST does not restrict data transfer to a single encoding like JSON, or communication to a particular protocol like HTTP. Fielding even indicated a RESTful architecture can be built on FTP. However, the protocol designated for interface between Components in Testbed 12 was HTTP, the underlying protocol of the World Wide Web.

Open geospatial REST also allows applications to select the output format most useful to their mission. For example, Resources can be represented in different encodings such as XML, RDF or JSON, depending on whether applications need complex analytics, linked data or simpler representations for web display.



Finally, Testbed 12 assessed that most of the pieces are in place for a REST WFS to support geospatial intelligence features as GeoJSON-only - without the need for a new profile document. Several issues must be addressed, however, this type of GeoJSON WFS would simplify application development and open the potential to begin moving from GML to linked features and observations in JSON.



With these key concepts you can begin to understand the basics of open geospatial Resources and REST. To better understand REST in the context of offering geospatial information and services you also need to know the Principles of REST and some of its advantages.

3.2. Advantages of REST

With an understanding of Resources, HTTP verbs, principles and levels, enterprises should be aware of some of the advantages of using REST to offer geospatial information and services. These advatages include -

1. Use of HTTP methods makes it 'easier' to build applications, saving time and money.
2. Lower barrier to entry, 'easier' for developers to use than SOAP.
3. Simple HTTP used to make calls rather than complex SOAP, POX.
4. Use of HTTP methods makes developing for database transactions 'easier'.
5. Requests sent with a simple URL templates to the server.
6. Easier navigation through hypermedia links.

3.3. Principles of REST

The term 'REST' was coined by Roy Fielding in Ph.D. dissertation in 2000. Since that document hundreds of systems architects have tried to establish one definition of a *REST Architecture* - usually resulting in an unfinished argument. This is because REST is not an architecuture. It's a set of principles that guide the development of a system that has resources, resource identifiers (e.g. URL), representations and connectors (e.g. clients, servers).

One interesting item to note is that the initial version of OGC Web Services were designed before Roy Fielding published his Ph.D. dissertation in 2000. So the answer to the question of "Why didn't OGC Web Services use Fielding's REST?" is... "Well, it didn't really exist."

It's also important to remember that many high-level principles of REST can seem rather Zen-like when you first read them. However, in Testbed 12 we limited things to HTTP, so things can start to fall into place.

With these considerations in mind, when considering REST for delivering geospatial data and services some high-level principles of REST include:

- REST does not restrict communication to a particular protocol.
- REST is not a web services pattern.

But before you think that 'Web Services' must be abandoned some other key points are ...

- The key abstraction of information in REST is a Resource.
- REST uses a Resource Identifier (e.g. URL) to identify the Resource involved in an interaction.
- REST connectors provide a generic interface for accessing and manipulating the value set of a Resource. Connectors are client and servers in Testbed 12.
- REST uses various connector types to encapsulate the activities of accessing Resources and transferring Resource Representations.
- A Representation is a sequence of bytes, plus Representation metadata to describe those bytes. REST doesn't say what the format of the Representation is.
- *REST does not restrict communication to a particular protocol, but it does constrain the interface between components, and hence the scope of interaction and implementation assumptions that might otherwise be made between components. For example, the Web's primary transfer Protocol is HTTP.*
- *The Protocol for interface between Components in Testbed 12 is HTTP.*

With these principles in mind, when trying to decide to use (or not use) REST to offer geospatial information and services it is important to remember that REST is not a 'one size fits all' proposition. There are different levels of REST ranging from not RESTful to what may be considered 'fully RESTful'. These were discussed in the previous section as Levels 0 to 3.

3.4. Key Terms

The following key terms and examples are presented in an order designed help the reader quickly

get a better understanding the basics of REST for geospatial information and services.

3.4.1. Representational State Transfer (REST)

REST is a simple method of discovering, accessing and updating geospatial and other information. There is no special software to install. A web browser, web application or a mobile app may be used to access the service directly - using standard HTTP methods (e.g. GET, PUT, POST, DELETE), resources and hypermedia controls.

3.4.2. Hypertext Transfer Protocol (HTTP)

HTTP is the communications protocol used on the World Wide Web by clients, like a web browser, and servers. RESTful systems often (but not always) communicate over Hypertext Transfer Protocol (HTTP).

3.4.3. HTTP Method

HTTP requests tell the server what the client wants and may include different 'methods' in the request such GET, PUT, POST and DELETE. The GET method retrieves information. The POST method sends information to be stored on the server. The PUT method sends a new copy of an existing information to the server. The DELETE method deletes information on a server. REST uses these HTTP methods to access and update 'resources' on the server.

3.4.4. URI

The unique identifier for a resource, structured in conformance with IETF RFC 2396, etc.

3.4.5. Resource

The key abstraction of information in REST is called a Resource. A Resource is simply the target of a HTTP method or hypertext reference. In an open geospatial REST architecture Resources can be maps, features, imagery, analytic processes, etc.

3.4.6. Base URL

This is a URL prefix that the client appends parameters to in order to arrive at a valid request URL, including a request URL for the Capabilities Resource. Typically, the Base URL contains a protocol identifier (e.g. http://) and a domain name (e.g. www.someserver.com).

3.4.7. Capabilities Resource

```
This is a description of the resources offered by a server.   The base, or root, URL
of the server resolves to the service metadata document, which means if you click it
in a web browser you should should a description of the services (the OGC Capabilities
document).
```

3.4.8. Hypermedia Controls

```
Hypermedia controls are links embedded within a resource representation. They tell a
client what state transition are available and how to perform them.
```

For the purposes of this guide, the definitions specified in the OWS Common Implementation Standard [OGC 06-121r9] also apply as needed.

3.4.9. Abbreviated Terms

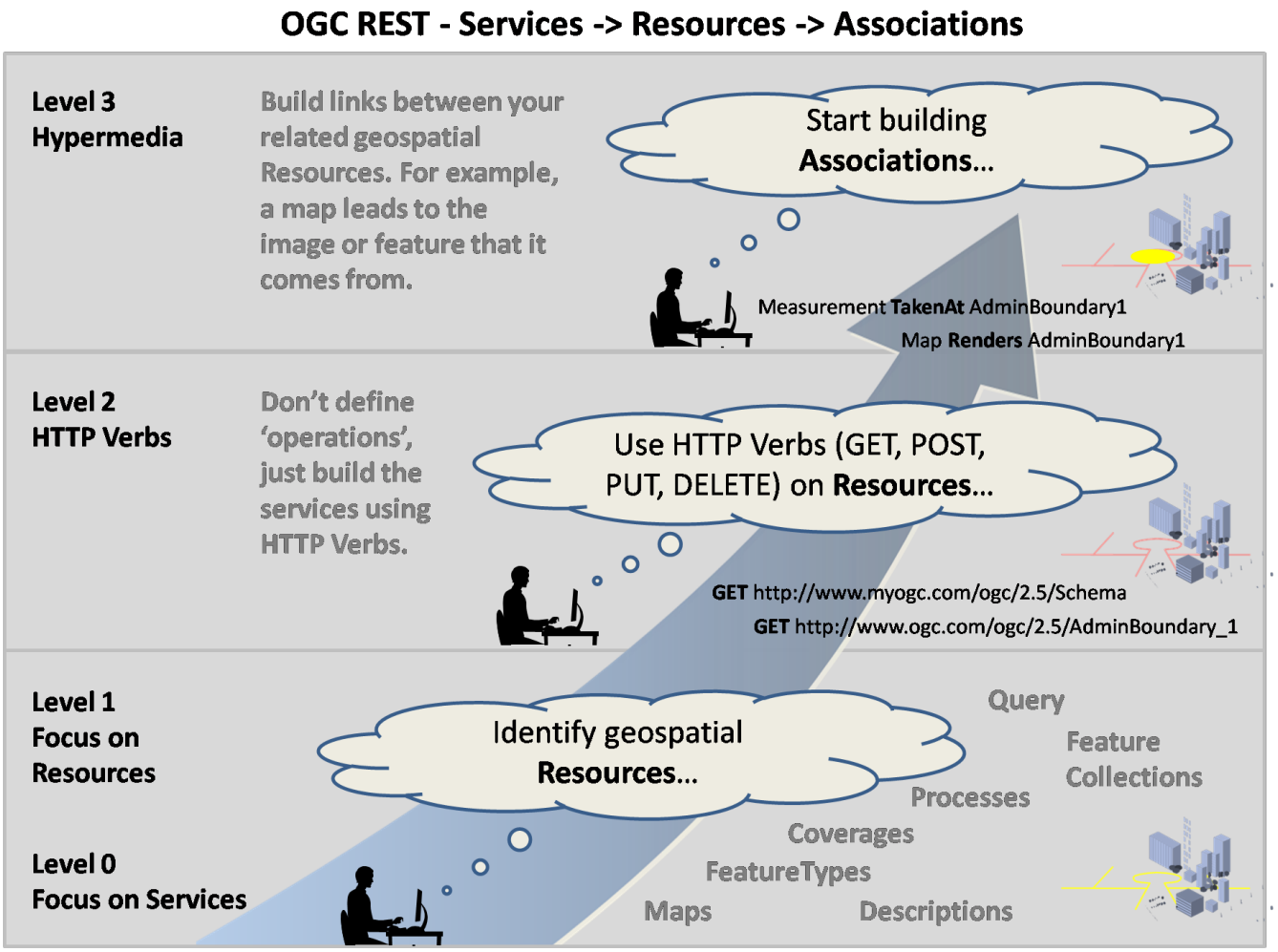
Frequently used abbreviated terms in this document include:

- HTMLHypertext Markup Language
- HTTPHypertext Transfer Protocol
- URLUniform Resource Locator
- APIApplication Programming Interface
- COTSCommercial Off The Shelf
- SOAPSimple Object Access Protocol
- WSDLWeb Services Definition Language
- JSON Javascript Object Notation
- XML Extensible Markup Language
- POX Plain Old XML
- RMM Richardsen Maturity Model

Chapter 4. Implementer Instructions

This section provides guidelines on REST resources and request examples from Testbed 12 to assist programmers/implementers.

From the perspective of an enterprise considering REST, the different levels offer a transition path that can reduce technical risks. As a reminder, here is the graphic from the previous section.



This path begins with using HTTP as the transport method for offering geospatial data and services, perhaps with a single HTTP method and XML descriptions. In the geospatial community this transition point is analogous to the first generation of OGC Web Services.

Next a geospatial enterprise begin to separate the API into Resources. In open geospatial REST Resources can be maps, features, imagery, analytic processes, etc. Then, HTTP methods GET, POST, PUT, DELETE are used to operate on the Resources.

The following sections provide some basic instructions to keep things simple and easy.

4.1. Use Nouns and Don't Use Verbs

You'll see this guidance in almost every REST guide, but it's there for a reason. For every open geospatial Resource use this guide.

Table 1. Noun Guide Table

Resource	GET	POST	PUT	DELETE
	Retrieve	Create	Update	Remove
/maps	Retrieves a list of maps	Creates a new map	Updates all maps	Deletes all maps
/maps/10	Retrieves a specific map	Not allowed (405)	Updates a specfic map	Deletes a specific map

Do not use verbs:

/getmaps

/createmaps

/deletemaps

At this point, most people familiar with geospatial technology will want to add more complexity. But for many geospatial enterprise APIs it's not needed.

Many thanks to Stefan Jauker for the inspiration for the above table. Note: The Resource 'maps' is used for illustrative purposes only.

4.2. Use Well-Known Resource Classes

Testbed 12 defined over 20 open geospatial Resource Classes (below). Geospatial Enterprises should use these well-known definitions their APIs.

Table 2. Testbed 12 Resource Classes

Resource	Definition
Capabilities	The complete service metadata document.
Tile	A rectangular pictorial representation of geographic data, often part of a set of such elements, covering a spatially contiguous extent and sharing similar information content and graphical styling, which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix.
FeatureInfo	Insert definition here from Wiki when there's time.
Schema	The complete application schema offered by the server.
Feature Type	A feature type (i.e. a named collection of features with the same schema)
Feature	A feature (i.e. a member of a feature type)

Resource	Definition
Feature Type Property	A named property from the schema of a feature type.
Feature Property	A named property from the schema of a feature.
Query	A complex query resource.
Transaction	A complex transaction resource.
Process	Detailed process description of a single process.
Process Collection	List of processes available.
JobCollection	List of jobs of a process.
Job	Representation of a job (execution of a process) containing status information.
Process Output Data	Resource containing the different process outputs inline or as reference.
Coverage	Full coverage in the format negotiated by the client and server through the proper HTTP Headers (e.g. Accept)
Coverage Description	Full metadata regarding one specific coverage in negotiated format.
Coverage Subset	A coverage derived on the fly from a subset operation applied to a persistent coverage. *The subset of a coverage is still a coverage.
Coverage Range	A coverage derived on the fly from a range subsetting operation applied to a persistent coverage *The range subset of a coverage is still a coverage.
	Insert definition here from Wiki when there's time.

4.3. Use HTTP Status Codes

The HTTP standard provides dozens of status codes to describe return values. These codes let client applications know when things go wrong, and also when they go right.

Not all HTTP Status Codes may be needed and the table below provides a short list of some useful for open geospatial REST. All HTTP Status Codes should be allowed.

NOTE The reader is encouraged to consult the individual bindings such as WFS, WCS, WMTS for specific status code use. The list below is for initial guidance only.

Table 3. HTTP Status Codes

HTTP Status Code	Meaning
200 OK	Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request, the response will contain an entity describing or containing the result of the action.
201 Created	The request has been fulfilled, resulting in the creation of a new resource.
202 Accepted	The request has been accepted for processing, but the processing has not been completed. The request might or might not be eventually acted upon, and may be disallowed when processing occurs.
203 Created	The request xxx been fulfilled, resulting in the creation of a new resource.
204 No Content	The server successfully processed the request and is not returning any content.
400 Bad Request	The server cannot or will not process the request due to an apparent client error.
401 Unauthorized	Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided.
403 Forbidden	The request was a valid request, but the server is refusing to respond to it.
404 Not Found	The requested resource could not be found but may be available in the future.
500 Internal Server Error	A generic error message, given when an unexpected condition was encountered and no more specific message is suitable. .

An example of the HTTP Status Code used after creating a feature Resource is shown below.

```
CLIENT                                                              SERVER
  |                                                                    |
  |   POST /mywfs/2.5/tx/xB7857n/ROADL_1M                  HTTP/1.1|
  |   Host: www.someserver.com                                         |
  |   ContentType: application/vnd.geo+json                            |
  |                                                                    |
  |   {"type":"Feature",                                               |
  |    "geometry":{"type":"LineString",                                |
  |                "coordinates":[[142.8419,50.6160],                  |
  |                               [142.8511,50.6137],                  |
  |                               [142.8510,50.6146],                  |
  |                               [142.8614,50.6139],                  |
  |                               [142.8861,50.6122],                  |
  |                               [142.9387,50.6170],                  |
  |                               [142.9427,50.6174],                  |
  |                               [142.9484,50.6184],                  |
  |                               [142.9534,50.6193],                  |
  |                               [142.9721,50.6228],                  |
  |                               [142.9914,50.6274],                  |
  |                               [143.0314,50.6370],                  |
  |                               [143.0655,50.6451],                  |
  |                               [143.0730,50.6469]]},                |
  |    "properties":{"gmlid":"CWFID.ROADL_1M.674",                     |
  |                  "id":166828,                                      |
  |                  "f_code":"AP030",                                 |
  |                  "acc":1,                                          |
  |                  "exs":28,                                         |
  |                  "med":2,                                          |
  |                  "rtt":15,                                         |
  |                  "tile_id":156,                                    |
  |                  "edg_id":462}}                                    |
  |------------------------------------------------------------------->|
  |                                                                    |
  |    HTTP/1.1 201 Created                                            |
  |    Location: /mywfs/2.5/tx/xB7857n/ROADL_1M/674                    |
  |<-------------------------------------------------------------------|
```

4.4. Use Well-Known URL Templates and Access Paths

This section provides examples of REST URL structures and Access Paths from Testbed 12.

4.4.1. WMTS - URL Templates and Access Paths

According to the WMTS Specification (OGC 07-057r7), the resources provided by a WMTS REST server are listed in the table below.

Table 4. Resources provided by the WMTS REST server

Resource Class	Description	Acccess Path
Capabilities	The complete service metadata document.	{WMTSBaseURL}/1.0.0/WMTSCapabilities.xml

Resource Class	Description	Acccess Path
Tile	A rectangular pictorial representation of geographic data which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix.	{WMTSBaseURL}/{TileMatrixSet}/{TileMatrix}/{TileRow}/{TileCol}.png
FeatureInfo	Information related to a particular pixel of a map that refers to the geographic data portrayed on that area.	{WMTSBaseURL}/{TileMatrixSet}/{TileMatrix}/{TileRow}/{TileCol}/{J}/{I}.xml

Notes:

1. {WMTSBaseURL}: The base URL of the WMTS REST endpoint.
2. {TileMatrixSet}: identifier of a set of tile matrizes.
3. {TileMatrix}: identifier to a specific tile matrix.
4. {TileRow}: identifier of a row of a specific tile matrix.
5. TileColumn}: identifier of a column of a specific tile matrix.
6. {J}: Pixel row index in a tile.
7. {I}: Pixel column index in a tile.

Besides the standard resources provided by a WMTS (i.e. tiles), if we define a TileJSON document of a layer to be a resource in and of itself, then the resource provided is a "TileJSON document".

The TileJSON document is defined by an open specification that can be found here: https://github.com/mapbox/tilejson-spec/tree/master/2.1.0. The document defines a very simple encoding for describing the availability of a WMTS Simple profile layer in Web Mercator. The WMTS Simple Profile specification (13-082r1) doesn't address TileJSON. However, as part of OGC Testbed 12 Cubewerx proposed a WMTS TileJSON extension. If/when this is written up as a WMTS profile specification it would be very similar in nature to the WMTS Simple Profile specification. The two profiles happen to dictate compatible tile matrix sets, so they could co-exist well.

The WMTS is a data portrayal service and as such, the only operation defined is the HTTP GET method which is used to retrieve the resources provided by the service (i.e. tiles and TileJSON documents).

A WMS Capabilities document provides links to the TileJSON document of each layer in two redundant ways:

1. By advertising a service-oriented GetTileJSON operation which takes SERVICE, VERSION, OPERATION=GetTileJSON and LAYER parameters
2. By advertising a "TileJSON" ResourceURL for each layer

A TileJSON document in turn provides links to a set of tiles by means of tile-URL templates (as defined by the TileJSON specification).

4.4.2. WFS - URL Templates and Access Paths

The resources that are provided by a WFS REST server are listed in the table below and include the capabilities document of the server, features, queries, transactions and certain metadata.

Table 5. Resources provided by the WFS REST server

Resource Class	Description	Acccess Path
Capabilities	The complete service metadata document.	{WfsRESTBaseURL}
Schema	The complete application schema offered by the server.	{schema URL}
Feature Type	A feature type (i.e. a named collection of features with the same schema)	{ftype schema URL}
Feature	A feature (i.e. a member of a feature type)	{feat URL}
Property of a feature type	A named property from the schema of a feature type	{ftype URL}/{prop} (See note 1)
Property of a feature	A named property from the schema of a feature	{feat URL}/{prop} (See note 1)
Query	A complex query resource	/query
Transaction	A complex transaction resource	/transaction

1. The property name is appended before any query parameters
2. {schema URL}: The URL to the application schemas that the service offers; this URL is specified in the capabilities document of the service using an atom:link element with rel="describedBy"
3. {ftype schema URL}: The URL to a schema document declaring the schema of the specified feature type; this URL is specified at the first nesting level within the wfs:FeatureType element in the capabilities document using an atom:link element with rel="describeBy"
4. {ftype URL}: The URL to the collection of feature 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 the rel="collection"
5. {feat URL}: The URL of a feature; it is specified in the response to a query within the feature container element (i.e. wfs:member element for an XML encoded response) using an atom:link element with rel="self"
6. {prop}: The name of a property of a feature or feature type; appended to the resource URL before any query parameters.

As discussed in the previous section, at Level 2 a geospatial enterprise may begin using the HTTP ‘verbs’ the same way they are used in HTTP itself. This means we may choose to not define unique ‘operations’ like GetFeature or Transaction (INSERT, UPDATE, DELETE) to access or modify Features. Instead, we use HTTP methods like GET, POST, PUT, DELETE. For example, the following table shows a sample of HTTP 'verbs' mapped to several of the traditional WFS 'operations' like GetFeature and

Transaction.

HTTP Method and Action	Access Paths	Equivalent WFS 'Operation'
GET (Retrieve)	/?... {schema URL} {ftype schema URL} {ftype URL}?... {query URL}?...	GetCapabilities DescribeFeatureType GetFeature GetFeature
POST (Create)	{ftype URL}?...	Transaction
PUT (Update)	(feat URL}?... {feat URL}/{prop}?...	Transaction Transaction
DELETE (Remove)	{feat URL}?... {feat URL}/{prop}?...	Transaction Transaction

It is important to note that emerging WFS drafts define four REST conformance classes:

- Simple Query
- Simple Transaction
- Complex Query
- Complex Transaction

The "Simple" conformance classes handle single feature type operations where the resource is the feature collection (i.e. the feature type) and the operations are GET, POST, PUT & DELETE.

All classes are optional so people who don't need or want complex query and transaction handling don't have to implement those Resources ... but if you want to support joins and atomic transactions you will need to.

Additional detailed REST WFS examples and information on WFS is provided in the 'Examples' section of this document. Overview examples are described in the 'REST and Open Geospatial Resources' section as the beginnning of this document.

4.4.3. WCS - URL Templates and Access Paths

The resources that are provided by a WCS REST server are listed in the table below and include the capabilities document of the server, coverages (including subsets) and corresponding metadata

Table 6. Resources provided by the WCS REST server

Resource Class	Description	Acccess Path	Example
Capabilities	The complete service metadata document. It contains a list of available extensions and all the available coverage resources	{WCSRestBaseUrl}/capabilities	http://ows.rasdaman.org/rasdaman/ows/capabilities

Resource Class	Description	Acccess Path	Example
Coverage	Full coverage in the format negotiated by the client and server through the proper HTTP Headers (e.g. Accept)	{WCSRestBaseUrl}/coverage/{coverageId}	http://ows.rasdaman.org/rasdaman/ows/coverage/AvgLandTemp
Coverage Description	Full metadata regarding one specific coverage in the negotiated format	{WCSRestBaseUrl}/coverage/{coverageId}/description	http://ows.rasdaman.org/rasdaman/ows/coverage/AvgLandTemp/description
Subset of a coverage	A coverage derived on the fly from a subset operation applied to a persistent coverage. *The subset of a coverage is still a coverage	{WCSRestBaseUrl}/coverage/{coverageId}/subset({subset})	http://ows.rasdaman.org/rasdaman/ows/coverage/AvgLandTemp/subset(Lat(10:20))
Range subset of a coverage	A coverage derived on the fly from a range subsetting operation applied to a persistent coverage *The range subset of a coverage is still a coverage	{WCSRestBaseUrl}/coverage/{coverageId}/rangesubset({subset})	http://ows.rasdaman.org/rasdaman/ows/coverage/AvgLandTemp/rangesubset(red,green)

Due to the protocol binding only GET operations can be applied on the resources mentioned in the section above. This follows the REST guidelines as all operations in WCS are idempotent and are mapped to concrete resources.

Since the first version of the protocol binding draft, the WCS-T extension of WCS (tranactional WCS) was adopted as a standard in OGC, which provides the first non-idempotent operation that a service serving coverages could implement. It is our opinion that based on this new addition to the ecosystem two new operations should be defined in a future draft of the protocol that allow PUT and PATCH operations on a coverage resource. The table below contains this proposal as well.

Table 7. Operations provided by the WCS REST server

HTTP Method	Acccess Path	Parameters	Description
GET	{wcsRESTBaseURL}	NA	Retrieval of Capabilites resource
PUT	{wcsRESTBaseURL}/coverage/MyCoverage	A valid coverage in one of the server supported formats	Creates a new coverage identified by the given url

HTTP Method	Acccess Path	Parameters	Description
PATCH	{wcsRESTBaseURL}/coverage/MyCoverage OR {wcsRESTBaseURL}/coverage/MyCoverage/subset(ansi("2012-01-01"))	A valid coverage in one of the server supported formats	Updates an existing coverage or a subset of it with the given input coverage

No direct associations are made between the resources as this is a Level 2 implementation. However a Level 3 implementation should aim to associate the Subset and RangeSubet of a Coverage with the original coverage in a meaningful way.

4.4.4. WPS - URL Templates and Access Paths

As there is not yet an official REST binding of the WPS available, the REST binding has been specified in Testbed 12. The resources that are provided by a WPS REST server are listed in the table below and include the capabilities document of the server, the list of processes available (ProcessCollection and Process), jobs (running processes) and outputs of processes.

Table 8. Resources provided by the WPS REST server

Resource Class	Description	Acccess Path	Example
Capabilities	The complete service metadata document.	{WpsRESTBaseURL}	http://geoprocessing.demo.52north.org:8080/wps-proxy
ProcessCollection	List of processes available	{WpsRESTBaseURL}/processes	http://geoprocessing.demo.52north.org:8080/wps-proxy/processes
Process	Detailed process description of a single process	{WpsRESTBaseURL}/processes/{process-id}	http://www.maps.bob/etopo2/default/WholeWorld_CRS_84/10m/1/3/86/132.xml
JobCollection	List of jobs of a process	{WpsRESTBaseURL}/processes/{processID}/jobs/{job-id}	NA
Job	Representation of a job (execution of a process) containing status information	{WpsRESTBaseURL}/processes/{processID}/jobs/{job-id}	NA
Process Output Data	Resource containing the different process outputs inline or as reference.	{WpsRESTBaseURL}/processes/{processID}/jobs/{job-id}/outputs	NA

1. {WpsRESTBaseURL}: The base URL of the WPS REST endpoint.
2. {process-id}: identifier of a process.
3. {job-id}: identifier to a job.

In general, the HTTP GET operation is used to provide access to the resources described above. However, in order to create a new job, the HTTP POST method is used to post a new job by sending a new job resource represented by an execute request to the server. This results in the operations listed in the table below.

Table 9. Operations on resources provided by the WPS REST server

HTTP Method	Acccess Path	Parameters	Description
GET	{WpsRESTBaseURL}	NA	Retrieval of Capabilities resource
GET	{WpsRESTBaseURL}/processes	NA	Retrieval of ProcessCollection resource
GET	{WpsRESTBaseURL}/processes/{process-id}	NA	Retrieval of a Process resource (process description)
GET	{WpsRESTBaseURL}/processes/{processID}/jobs	NA	Retrieval of list of jobs of a specific process (JobCollection)
GET	{WpsRESTBaseURL}/processes/{processID}/jobs/{job-id}	NA	Retrieval of a single Job resource
POST	{WpsRESTBaseURL}/processes/{processID}/jobs	Execute request (contained in body)	Execution of a process

Notes 1. {WpsRESTBaseURL}: The base URL of the WPS REST endpoint 2. {process-id}: identifier of a process. 3. {job-id}: identifier to a job.

As stated above in the listing of resources, the basic resources managed by the WPS REST server are processes, jobs, outputs and its corresponding collections. The Capabilities has an association to the ProcessCollection that aggregates the single Processes offered by the Web Processing Server. The process has an association to the collection of jobs (JobCollection) that aggregates single jobs, i.e. instances that run a certain process.

4.4.5. WMS - URL Templates and Access Paths

The WMS work item for Testbed 12 involved investigating and implementing the use of GeoJSON as an output format for the WMS GetFeatureInfo operation as described in OGC 15-053, “Testbed 11 Implementing JSON/GeoJSON in an OGC Standard Engineering Report”. Specifically, this work item was concerned with recommendation 28 from OGC 15-053 which proposed including an encoding for GetFeatureInfo responses based on GeoJSON. Recommendation 28 further proposed replacing the geometry part by a marker of the position of the query and the position of the returned feature. If the returned objects correspond to simple features, an identifier is to be included in the response that allows recovering the geometry using an additional WFS query. The Testbed 12 RFP proposed using the proposed WMS 1.4 draft specification as the base service standard for this work. The state of the WMS 1.4 draft specification, however, is such that it would not be possible to implement a

service with the WMS 1.4 API within the time and resource allocated for this work item in Testbed 12. Instead, the existing and more stable WMS 1.3 (OGC 06-042) server was proposed and used for this work item.

We can infer from the purpose of the GetFeatureInfo operation that the relevant resource is a “feature” and that a GeoJSON response would be considered a representation of that feature.

NOTE	The current draft WMS 1.4 specification (which seems to deprecate an existing draft WMS 2.0 version based on the modification dates of the documents) discusses REST only in a peripheral way and not related to the GetFeatureInfo operation at all.

The WMS is a data portrayal service and as such, the only HTTP method used is the GET method which is used to retrieve the resources provided by the service (i.e. maps and feature info).

4.5. JSON Examples

This section presents Testbed 12 experience using open geospatial REST and JSON.

4.5.1. JSON and WFS

This section presents Testbed 12 experience using open geospatial REST with WFS and JSON. The resulting JSON WFS REST API was able to represent real world phenomena as open geospatial features. The features were deployed as open geospatial Resources that can be accessed and updated using the language of the World Wide Web.

This section consists of the following sub-sections:

- Background - Open Geospatial REST, WFS, NAS GML, GeoJSON
- GeoJSON WFS REST Examples
- Lessons Learned

Background

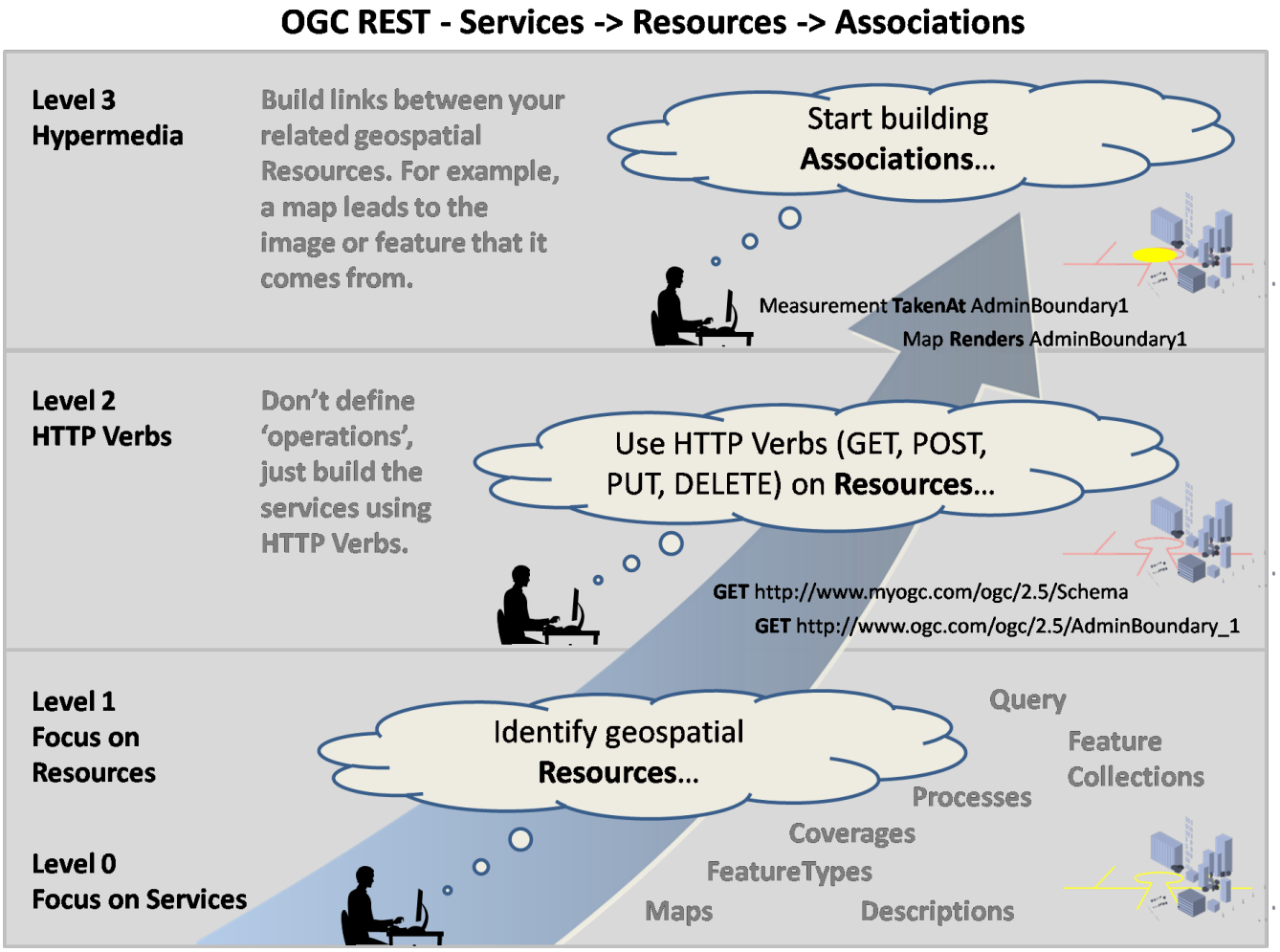
Technology Integration Experiments in this segment of Testbed 12 by The Carbon Project brought together four aspects of Testbed 12 -

- Open Geospatial REST
- WFS
- NAS GML
- GeoJSON

Open Geospatial REST

For the purposes of Testbed 12 we limited REST to the Web (i.e. HTTP) - and then broke it down into four Levels. These Levels are derived for the Richardson Maturity Model (RMM). The four Levels progress from Services to a focus on geospatial Resources, HTTP Verbs and linking Resources. A simplified representation of this Services → Resources → Associations progression is presented in

the graphic below. It's important to recognize that these Levels aren't discrete and exclusive – they build upon each other to help geospatial interoperability.



Starting at Level 0 the focus is on making requests to 'Services' using HTTP as a transport system for our remote interactions, often with XML. Level 0 represents the current state of many OGC Web Services. At Level 1 an enterprise can start to identify open geospatial Resources and then make requests to the Resources. An open geospatial Resource is geospatial information, like a feature, that can be identified by a URL.

At Level 2, a geospatial enterprise may begin using the HTTP 'verbs' the same way they are used in HTTP itself. This means we don't define unique 'Operations' like INSERT, UPDATE, DELETE to access or modify Features. Instead, we use HTTP methods like GET, POST, PUT, DELETE. Using these HTTP methods can make many things easier for API developers and client applications. Where possible, WFS REST strives to HTTP verbs for each action against a Resource. For example:

Table 10. HTTP Methods Guide Table

Resource	GET	POST	PUT	DELETE
	Retrieve	Create	Update	Remove
/featuretypes	Retrieves a list of featuretypes	Creates new featuretypes	Updates all featuretypes	Not allowed (405)

Resource	GET	POST	PUT	DELETE
/featuretypes/AdministrativeSubdivision	Retrieves specific featuretypes	Creates specfic featuretypes	Updates specfic featuretypes	Deletes specific featuretypes

With Resources defined and HTTP verbs in use, at Level 3 we begin to focus is on hypermedia and Associations. That means we can start building links between related open geospatial Resources. For example, hypermedia links may be used to link metadata to different information models or schemas.

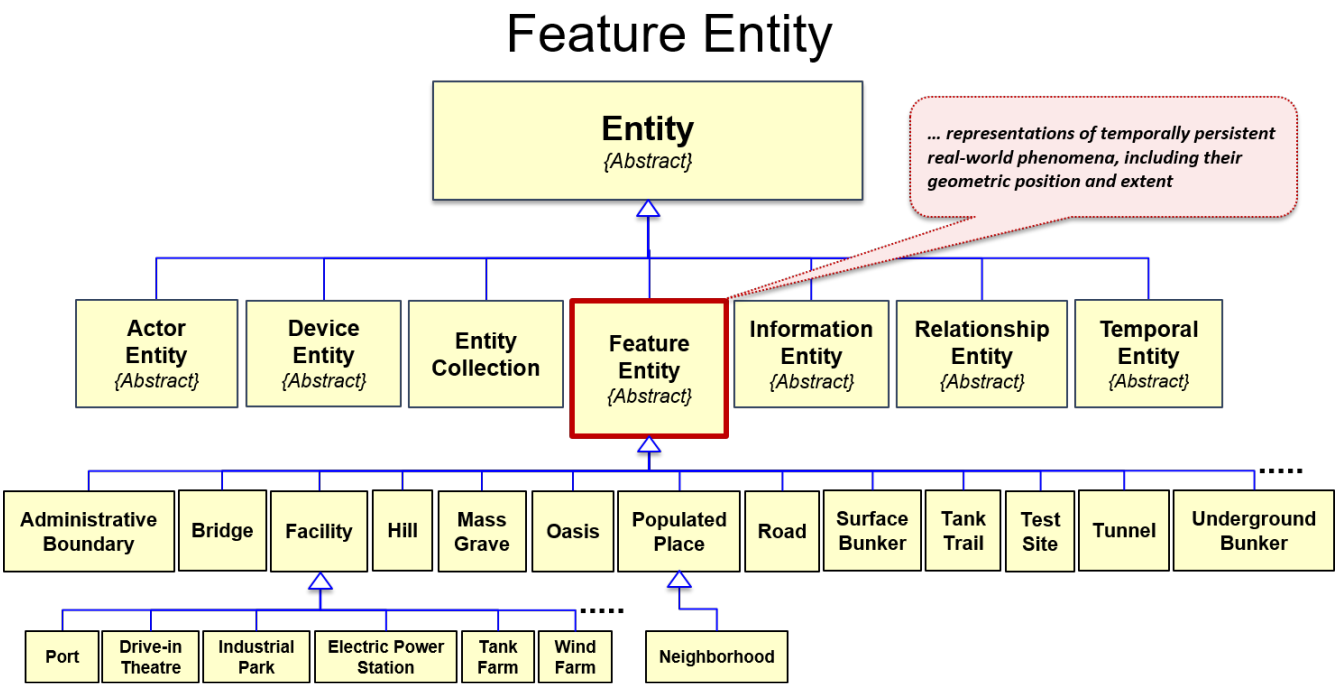
WFS

The OGC Web Feature Service (WFS) Implementation Specification allows a client to retrieve geospatial data encoded in Geography Markup Language (GML) and other formats. The specification defines operations for data access and manipulation operations on geographic features, using HTTP as the distributed computing platform. Via these interfaces, a Web user or service can combine, use and manage geodata — the feature information behind a map image. Relevant WFS operations for this segment of Testbed 12 include -

- Query operations to allow features or values of feature properties to be retrieved based upon constraints, defined by the client, on feature properties.
- Transaction operations allow features to be created, changed, replaced and deleted from the underlying data store.

NAS GML

NSG Application Schema (NAS) is a model that defines the GEOINT exchange semantics for the US National System for Geospatial-Intelligence (NSG). For Testbed 12, NAS GML was provided to test the ability of interoperable components to support GML encodings of the model, in particular the Feature Entity portion of NAS -



For Testbed 12 and NAS GML, key concepts are Features and Observations. An Observation is a Feature and Observations identify correlations between data. Observation may be generated by manual recognation of certain patterns or through automated analytics. Correlations are statements of connections, but don't indicate meaning. An example is shown below.

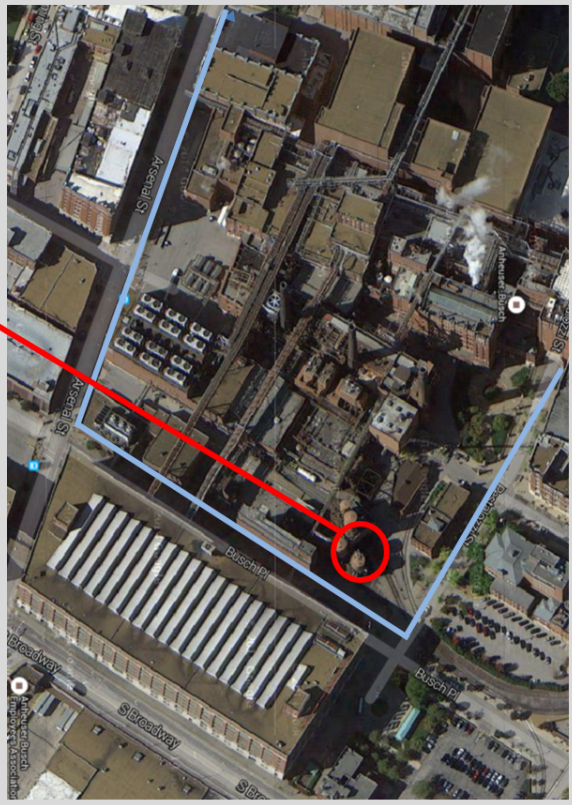
Example - Observations are Features, and Features may be Linked

On July 15, 2015, Analyst Nick Rivers assessed that a change has taken place at a known beverage production facility (Non-Alcoholic Beer Brewery).

Mr. River's observation of satellite image source (Google Earth image captured on June 1, 2015) indicates the presence of a newly-constructed beer-precursor storage tank within the existing facility footprint (existing feature).

The new feature (water tank) is recorded as a new facility-contained structure associated with the existing feature and the content database is updated.

The new feature, its attributes and relationship to the facility, and the collection information (sources, methods, assessments) are recorded, including a security marking of FOUO.



Given this challenge, the crucial step for Testbed 12 is to begin moving from individual features to linked features. The association describes the relationship between features and is thus the binding connection, linking objects with each other to handle challenges like the one discussed above.

For Testbed 12 Adminstrative Boundary features in NAS GML were provided to participants for testing an encoding of NSA GML as GeoJSON delivered by a WFS.

GeoJSON

GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON). It defines several types of JSON objects and the manner in which they are combined to represent data about geographic features, their properties, and their spatial extents. GeoJSON uses a geographic coordinate reference system, World Geodetic System 1984, and units of decimal degrees.

GeoJSON objects may represent a region of space (a Geometry), a spatially-bounded entity (a Feature), or a list of features (a Feature Collection). GeoJSON supports the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection. Features in GeoJSON contain a geometry object and additional properties, and a Feature Collection that contains a list of features.

GeoJSON features are this combination of geometry and properties:

```
{

 "type": "Feature",
 "geometry": {
  "type": "Point"'
  "coordinates": [0, 0]

 },

 "properties": {
  "name": "admin boundary"

 }
}
```

The properties attached to a feature can be any kind of JSON object.

GeoJSON WFS REST Examples

In Testbed 12 Open Geospatial REST, WFS, NAS GML and GeoJSON were integrated to deploy a GeoJSON WFS REST.

- **GML**

http://ows12.azurewebsites.net/wfs/featuretypes/AdministrativeSubdivision

- **JSON**

http://ows12.azurewebsites.net/wfs/featuretypes/AdministrativeSubdivision?outputFormat=json

Note: In most JSON WFS deployments there may be only JSON output format and the request would be directed to the Administrative Subvision feature Resource.

Request GeoJSON

Simple Query

```
GET /featuretypes/AdministrativeSubdivision?outputFormat=json
HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type:
application/vnd.geo+json

{ "type": "FeatureCollection", "features": [{"type":"Feature","_id":"1",,"geometry":{
"type": "Polygon", "coordinates": [ [ [-122.438548585768, 37.7527347148009], [-
122.438712218419, 37.7542478576584], [-122.437694850195, 37.75428723271], [-
```

```
122.437588133248, 37.7528134664936], [-122.438548585768, 37.7527347148009] ] ]
},"properties":{
  "nas:uniqueEntityIdentifier": "UUID-1-132904",
  "nas:place": {
    "nas:FeaturePlaceInfo": {
      "nas:uniqueEntityIdentifier": "UUID-1-132904",
      "nas:place": {}
    }
  },
  "nas:restriction": {
    "nas:RestrictionInfo": {
      "nas:uniqueEntityIdentifier": "UUID-1-here",
      "nas:commercialCopyrightNotice": {
        "nas:TextLexUnconMeta": {
          "nas:valueOrReason": "Copyright National Geospatial-Intelligence Agency"
        }
      },
      "nas:securityAccessGroup": {
        "nas:RestrictionInfoSecurityAttributesGroupMeta": {
          "nas:valueOrReason": {
            "attributes": [
              {
                "icism:classification": "U"
              },
              {
                "icism:ownerProducer": "USA"
              }
            ],
            "value": null
          }
        }
      }
    }
  },
  "nas:area": {
    "nas:RealNonNegMeta": {
      "nas:restriction": {
        "nas:RestrictionInfo": {
          "nas:uniqueEntityIdentifier": "UUID-1-here",
          "nas:commercialCopyrightNotice": {
            "nas:TextLexUnconMeta": {
              "nas:valueOrReason": "Copyright National Geospatial-Intelligence Agency"
            }
          },
          "nas:securityAttributesGroup": {
            "nas:RestrictionInfoSecurityAttributesGroupMeta": {
              "nas:valueOrReason": {
                "attributes": [
                  {
                    "icism:classification": "U"
                  },
```

```
                {
                  "icism:ownerProducer": "USA"
                }
              ],
              "value": null
            }
          }
        }
      }
    }
  },
  "nas:bgnAdminLevel": {
    "nas:AdministrativeSubdivisionBgnAdminLevelCodeMeta": {
      "nas:valueOrReason":
"http://api.nsgreg.nga.mil/codelist/BgnAdminLevel/firstOrder"
    }
  },
  "nas:designation": {
    "nas:AdminSubdivisionDesig": {
      "nas:uniqueEntityIdentifier": "UUID-1-132904",
      "nas:gencPreferredName": {
        "nas:TextLexUnconMeta": {
          "nas:valueOrReason": "California"
        }
      },
      "nas:gencShortUrnBasedIdentifier": {
        "nas:GencShortUrnBasedIdentifierTextMeta": {
          "nas:valueOrReason": "ge:GENC:3:3-3:US-CA"
        }
      }
    }
  },
  "nas:principalSubdivisionOf": {
    "nas:GeopoliticalEntity": {
      "nas:uniqueEntityIdentifier": "ge:GENC:3:3-3:USA",
      "nas:boundary": null,
      "nas:designation": {
        "nas:GeopoliticalEntityDesig": {
          "nas:uniqueEntityIdentifier": "UUID-1-132904",
          "nas:gencShortUrnBasedIdentifier": {
            "nas:GencShortUrnBasedIdentifierTextMeta": {
              "nas:valueOrReason": "ge:GENC:3:3-3:US-CA"
            }
          }
        }
      }
    }
  }
}}]}
```

In the example above, the response is a combination of geometry and properties, with the geometry type of Polygon. Since this is GeoJSON only EPSG code 4326 should be returned.

Create GeoJSON FeatureType

Simple Transaction

```
POST /featuretypes/AdministrativeSubdivision?outputFormat=json
HTTP/1.1
Content-Type:
application/vnd.geo+json

{
   "type":"Feature",
   "geometry":{
      "type":"Polygon",
      "coordinates":[
         [
            [
               -122.478531156081,
               37.7394760665792
            ],
            [
               -122.478795146849,
               37.7409374031664
            ],
            [
               -122.477965461578,
               37.7409374031664
            ],
            [
               -122.477739183777,
               37.7394760665792
            ],
            [
               -122.478531156081,
               37.7394760665792
            ]
         ]
      ]
   },
   "properties":{
      "nas:uniqueEntityIdentifier":"UUID-1-132904",
      "nas:place":{
         "nas:FeaturePlaceInfo":{
            "nas:uniqueEntityIdentifier":"UUID-1-132904",
            "nas:place":{
               "nas:featureEntity":null
            }
         }
      },
```

```
      "nas:restriction":{
         "nas:RestrictionInfo":{
            "nas:uniqueEntityIdentifier":"UUID-1-here",
            "nas:commercialCopyrightNotice":{
               "nas:TextLexUnconMeta":{
                  "nas:valueOrReason":"Copyright National Geospatial-Intelligence
Agency"
               }
            },
            "nas:securityAttributesGroup":{
               "nas:RestrictionInfoSecurityAttributesGroupMeta":{
                  "nas:valueOrReason":null
               }
            }
         }
      },
      "nas:area":{
         "nas:RealNonNegMeta":{
            "nas:valueOrReason":"415836540000",
            "nas:restriction":{
               "nas:RestrictionInfo":{
                  "nas:uniqueEntityIdentifier":"UUID-1-here",
                  "nas:commercialCopyrightNotice":{
                     "nas:TextLexUnconMeta":{
                        "nas:valueOrReason":"Copyright National Geospatial-
Intelligence Agency"
                     }
                  },
                  "nas:securityAttributesGroup":{
                     "nas:RestrictionInfoSecurityAttributesGroupMeta":{
                        "nas:valueOrReason":null
                     }
                  }
               }
            }
         }
      },
      "nas:bgnAdminLevel":{
         "nas:AdministrativeSubdivisionBgnAdminLevelCodeMeta":{

"nas:valueOrReason":"http://api.nsgreg.nga.mil/codelist/BgnAdminLevel/firstOrder"
         }
      },
      "nas:designation":{
         "nas:AdminSubdivisionDesig":{
            "nas:uniqueEntityIdentifier":"UUID-1-132904",
            "nas:gencPreferredName":{
               "nas:TextLexUnconMeta":{
                  "nas:valueOrReason":"California"
               }
            },
```

```
            "nas:gencShortUrnBasedIdentifier":{
               "nas:GencShortUrnBasedIdentifierTextMeta":{
                  "nas:valueOrReason":"ge:GENC:3:3-3:US-CA"
               }
            }
         }
      },
      "nas:principalSubdivisionOf":{
         "nas:GeopoliticalEntity":{
            "nas:uniqueEntityIdentifier":"ge:GENC:3:3-3:USA",
            "nas:boundary":null,
            "nas:designation":{
               "nas:GeopoliticalEntityDesig":{
                  "nas:uniqueEntityIdentifier":"UUID-1-132904",
                  "nas:gencShortUrnBasedIdentifier":{
                     "nas:GencShortUrnBasedIdentifierTextMeta":{
                        "nas:valueOrReason":"ge:GENC:3:3-3:US-CA"
                     }
                  }
               }
            }
         }
      }
   }
}
```

```
HTTP/1.1 201 Created
```

Additional examples such as changing the value of a property would use PUT method etc.

Lessons Learned

Based on the experiences of Testbed 12, a GeoJSON WFS may have significant potential to represent real world phenomena as open geospatial REST Resources that can be accessed and updated using the language of the World Wide Web.

Most of the pieces are almost in place for WFS to support NAS GML as GeoJSON right 'out of the box' using the REST binding, without the need for a profile document. However, the following issues must be considered -

- No Mandatory GML Output
- Bindings
- Capabilities (or not Capabilities)
- Feature Schema
- Attributes
- Associations and Namespaces

Lesson 1: No Mandatory GML output

The first requirement is that WFS not mandate GML. This is not currently the case, but there is a change proposal (OGC 13-061) that is requesting GML not be mandatory any longer and that WFS clients and servers negotiate a mutually agreeable response format. Removing GML as a mandatory output format would allow a WFS to support JSON without also having to implement NAS GML at the same time. This will greatly lower the bar for implementers and promote faster adoption.

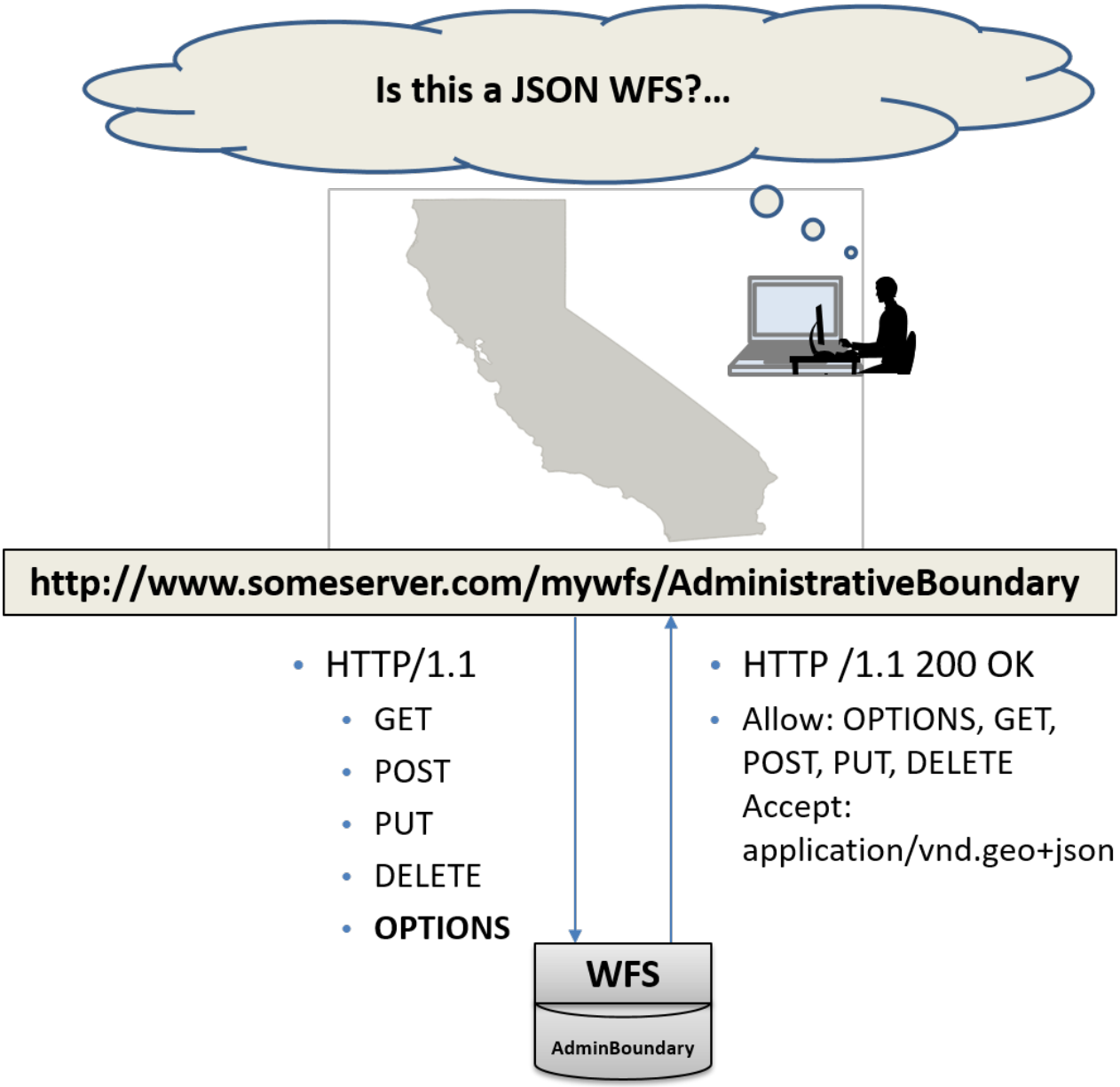
Lesson 2: Bindings

The WFS supports four bindings: KVP-GET, XML-POST, SOAP and REST. A JSON-only response is likely possible with all these binding, but it only really makes sense for the KVP-GET and REST binding. The other two bindings are tied to XML and mixing XML and JSON is awkward and probably too much for a client application to easily deal with. So it is reasonable to assert that handling JSON with a WFS only makes sense for the KVP-GET and REST bindings (keeping in mind that the KVP-GET binding is a read-only binding — i.e. no transactions).

Lesson 3: Capabilities (or not Capabilities)

For a client to commence communicating with an OGC web service it usually must retrieve and understand the service's Capabilities document. The current problem is that the only encoding defined for the Capabilities Resource is XML, which would be awkward in a JSON-only interaction chain such as the one discussed for NAS JSON. The ideal solution is that OWS Common SWG define a JSON encoding for Capabilities Resources, but this was not available during Testbed 12 and is probably not forthcoming in the near future.

However, there is a potential solution to this challenge. In Testbed 12 The Carbon Project and Cubewerx assessed that the WFS REST binding somewhat mitigates the necessity of having a Capabilities Resource altogether by using the HTTP OPTIONS method. The HTTP OPTIONS method is used to get available options for Resources (i.e. what you can do with Resources). For example, given the URL for a Feature Resource a client can make an OPTIONS call to learn which methods and representations are supported. For example:



We can see that for this Feature Resource (i.e. AdministrativeSubdivision) the API supports all the HTTP methods and the GeoJSON format. In many situations this may be simpler than relying completely on the capabilities document. Instread, I may simply give you the URL of a Feature Resource and your client can figure out if GeoJSON is a supported format. No XML is required with this method, simplying development.

Lesson 4: Feature Schema

For the KVP-GET, XML-POST and SOAP bindings the DescribeFeatureType operation may be used to get the schema of a feature. For the WFS REST binding serving , hypermedia controls (rel="describedBy") can be used to retrieve the schema in any number of schema description languages. This control can appear in a number of places including the capabilities document of the server and in the response to a GetFeature request. In many situations this may be simpler than relying completely on the the DescribeFeatureType operation. Again, no XML is required as the schema may be in JSON.

Lesson 5: Attributes

A particular challenge identified in Testbed 12 was NAS GML attribute handling. For example, the

following GML indicates the security tagging (classification) of a geospatial Feature Entity - Administrative Boundary -

```
<nas:valueOrReason
icism:classification="U" icism:ownerProducer="USA"
></nas:valueOrReason>
```

To represent the security tags of the Administrative Boundary Feature Entity in JSON we enclose multiple objects in square brackets, signifying an array. For example, the attributes representing 'icism classification' were as follows -

```
"nas:valueOrReason": {
 "attributes": [
  {
    "icism:classification": "U"
  },
  {
    "icism:ownerProducer": "USA"
  }
  ],
   "value": null
  }
```

This may not represent the 'best' solution, but since there is little OGC guidance methodology this method may serve as a foundation for consideration.

Lesson 6: Associations and Namespaces

As shown in the example above, an Observation is a Feature and there should be a path to move from individual features to linked features. The association describes the relationship between features and is thus the binding connection, linking objects with each other to handle challenges like the one discussed above. To represent associations hypermedia controls may be used to link one feature to another. Namespaces in the NAS GML may also be represented in a similar manner, where the link to a namespace is a URL and the Namespace is JSON.

4.5.2. JSON and WMTS

The TileJSON-generating WMTS server was deployed in Testbed 12 at the following URL:

http://tb12.cubewerx.com/a042/cubeserv/default/wmts/1.0.0/WMTSCapabilities.xml

The following is a RESTful example of accessing a TileJSON document from this server using the HTTP GET method:

```
http://tb12.cubewerx.com/a042/cubeserv/default/wmts/1.0.0/tileJSON/National_Land_Cover
.National_Land_Cover
```

The following TileJSON document is retrieved by this URL (NOTE: Normally, packed JSON output is generated from by the A042 server but the output has been wrapped in this example to make it easier to read):

```
{"tilejson":"2.1.0",
    "name":"National Land Cover",
    "tiles":["http://tb12-1.cubewerx.com/a042/OpenImageMap/tilesets/USGS/
              National_Land_Cover/National_Land_Cover/default/
              smerc/{z}/{y}/{x}.jop",
             "http://tb12-2.cubewerx.com/a042/OpenImageMap/tilesets/USGS/
              National_Land_Cover/National_Land_Cover/default/
              smerc/{z}/{y}/{x}.jop",
             "http://tb12-3.cubewerx.com/a042/OpenImageMap/tilesets/USGS/
              National_Land_Cover/National_Land_Cover/default/
              smerc/{z}/{y}/{x}.jop",
             "http://tb12-4.cubewerx.com/a042/OpenImageMap/tilesets/USGS/
              National_Land_Cover/National_Land_Cover/default/
              smerc/{z}/{y}/{x}.jop"],
    "bounds":[-2493045,1317885,-1713555,2497245]
   }
```

Lessons Learned

This exercise showed that the WMTS specification can be easily extended to support other tile-access strategies.

TileJSON essentially does what WMTS does, but in a more restrictive way, allowing only the Spherical Mercator coordinate system, a single layer, a single style, no extra dimensions, no feature-info querying, etc.. This makes TileJSON clients a bit easier to implement than WMTS clients (assuming that the restrictions of TileJSON are acceptable).

Adding TileJSON support to a WMTS could help allow a TileJSON client to use the WMTS. However, it's not a perfect plug-in solution for TileJSON clients, because the WMTS capabilities document still needs to be parsed (either by a human or automatically) in order to determine the TileJSON URLs.

4.5.3. JSON and WPS

The 52°North WPS REST API was realized as a proxy to be able to test it with different WPS implementations during the specification process. The proxy was written in Java using the Spring framework and jackson libraries for XML and JSON handling. The WPS requests and responses are handled using the 52°North WPS 2.0 XMLBeans. The underlying WPS was a 52°North WPS 2.0 instance, wrapping the Hootenanny conflation software.

In the following, example requests and responses for the different operations and resources will be shown.

Example of HTTP GET request for retrieving JSON WPS Capabilities.

```
http://tb12.dev.52north.org/wps-rest
```

Example of WPS Capabilities encoded as JSON.

```
{
    "Capabilities": {
        "ServiceIdentification": {
            "Title": "52°North WPS 4.0.0-SNAPSHOT",
            "Abstract": "Service based on the 52°North implementation of WPS 1.0.0",
            "ServiceType": "WPS",
            "ServiceTypeVersion": ["1.0.0",
            "2.0.0"],
            "Fees": "NONE",
            "AccessConstraints": "NONE"
        },
        "ServiceProvider": {
            "ProviderName": "52North",
            "ProviderSite": {
                "HRef": "http://www.52north.org/"
            },
            "ServiceContact": {
                "IndividualName": "Your name",
                "ContactInfo": {

                }
            }
        },
        "Contents": {
            "ProcessSummaries": [{
                "identifier": "testbed12.fo.DouglasPeuckerAlgorithm",
                "title": "testbed12.fo.DouglasPeuckerAlgorithm",
                "_processVersion": "1.0.0",
                "_jobControlOptions": "sync-execute",
                "_outputTransmission": "value",
                "url": "http://tb12.dev.52north.org:80/wps-
rest/processes/testbed12.fo.DouglasPeuckerAlgorithm"
            },...
            ]
        },
        "_service": "WPS",
        "_version": "2.0.0"
    }
}
```

The process summaries in the contents-section contain links to the process description of the respective process. The standalone list of processes can be requested as follows:

Example of HTTP GET request for retrieving the list of offered processes encoded as JSON.

```
http://tb12.dev.52north.org/wps-rest/processes
```

Example of WPS Capabilities encoded as JSON.

```
{
  "ProcessSummaries": [
    {
      "identifier": "testbed12.fo.DouglasPeuckerAlgorithm",
      "title": "testbed12.fo.DouglasPeuckerAlgorithm",
      "_processVersion": "1.0.0",
      "_jobControlOptions": "sync-execute",
      "_outputTransmission": "value",
      "url": "http://tb12.dev.52north.org:80/wps-
rest/processes/testbed12.fo.DouglasPeuckerAlgorithm"
    },...
    ]
}
```

Example of HTTP GET request for retrieving the process description of a process encoded in JSON. (NOTE: Request has been line-wrapped for easier reading).

```
http://tb12.dev.52north.org/wps-rest/processes/
testbed12.fo.DouglasPeuckerAlgorithm
```

Example of WPS Capabilities encoded as JSON.(NOTE: Some inputs and formats have been left out for easier reading).

```
{
  "ProcessOffering": {
    "Process": {
      "Title": "Hootenanny Conflation Process",
      "Identifier": "testbed12.lsa.HootenannyConflation",
      "Input": [
        {
          "Title": "INPUT1",
          "Identifier": "INPUT1",
          "ComplexData": {
            "Format": [
              {
                "_default": "true",
                "_mimeType": "application/x-zipped-shp"
              },...
            ]
          },
          "_minOccurs": "1",
          "_maxOccurs": "1"
        },
```

```
        {
          "Title": "INPUT1_TRANSLATION",
          "Identifier": "INPUT1_TRANSLATION",
          "ComplexData": {
            "Format": [
              {
                "_default": "true",
                "_mimeType": "text/x-script.phyton"
              },
              {
                "_default": "false",
                "_mimeType": "text/plain"
              }
            ]
          },
          "_minOccurs": "0",
          "_maxOccurs": "1"
        },
        {
          "Title": "INPUT2",
          "Identifier": "INPUT2",
          "ComplexData": {
            "Format": [
              {
                "_default": "true",
                "_mimeType": "application/x-openstreetmap+xml"
              }...
            ]
          },
          "_minOccurs": "1",
          "_maxOccurs": "1"
        },
        {
          "Title": "CONFLATION_TYPE",
          "Identifier": "CONFLATION_TYPE",
          "LiteralData": {
            "Format": [
              {
                "_default": "true",
                "_mimeType": "text/plain"
              },
              {
                "_default": "false",
                "_mimeType": "text/xml"
              }
            ],
            "LiteralDataDomain": [
              {
                "AnyValue": null,
                "DataType": {
                  "_reference": "xs:string"
```

```
              }
            }
          ]
        },
        "_minOccurs": "0",
        "_maxOccurs": "1"
      },...
    ],
    "Output": [
      {
        "Title": "CONFLATION_OUTPUT",
        "Identifier": "CONFLATION_OUTPUT",
        "ComplexData": {
          "Format": [
            {
              "_default": "true",
              "_mimeType": "application/x-zipped-shp"
            },...
          ]
        }
      },
      {
        "Title": "CONFLATION_REPORT",
        "Identifier": "CONFLATION_REPORT",
        "ComplexData": {
          "Format": [
            {
              "_default": "true",
              "_mimeType": "text/plain"
            }
          ]
        }
      }
    ]
  },
  "_processVersion": "1.0.0",
  "_jobControlOptions": "sync-execute async-execute",
  "_outputTransmission": "value reference",
  "execute-url": "http://tb12.dev.52north.org:80/wps-
rest/processes/testbed12.lsa.HootenannyConflation/jobs"
 }
}
```


Example of HTTP GET request for getting a list of jobs of a process. (NOTE: Request has been line-wrapped for easier reading).

```
http://tb12.dev.52north.org/wps-rest/processes/
testbed12.fo.DouglasPeuckerAlgorithm/jobs
```

Example of a list of jobs for a process encoded as JSON.

```
{
  "Jobs": [
    "1317c058-cb4d-4ab4-ad21-b78e51229a17",
    "1319d2fc-cac8-4e8d-8039-2c511f55a9d3"
  ]
}
```

Example of HTTP POST request for executing a process. (NOTE: Request has been line-wrapped for easier reading).

```
http://tb12.dev.52north.org/wps-rest/processes/
testbed12.fo.DouglasPeuckerAlgorithm/jobs
```

By default, the proces will be executed asynchronously. If the process supports synchronous execution, this can be achieved by appending the following URL-parameter:

Example of HTTP POST request for synchronously executing a process. (NOTE: Request has been line-wrapped for easier reading).

```
http://tb12.dev.52north.org/wps-rest/processes/
testbed12.fo.DouglasPeuckerAlgorithm/jobs?sync-execute=true
```

Example of WPS Execute request encoded as JSON.

```
{
    "Execute": {
        "Identifier": "testbed12.lsa.HootenannyConflation",
        "Input": [
            {
            "Reference": {
                    "_mimeType": "application/x-zipped-shp",
                    "_href":
"http://geoprocessing.demo.52north.org:8080/data/Trans_RoadSegment-aoi.zip"
            },
            "_id": "INPUT1"
            },
            {
            "Reference": {
                    "_mimeType": "text/x-script.phyton",
                    "_href":
"http://geoprocessing.demo.52north.org:8080/data/TNM_Roads.py"
            },
            "_id": "INPUT1_TRANSLATION"
            },
            {
            "Reference": {
                    "_mimeType": "application/x-openstreetmap+xml",
                    "_href":
"http://geoprocessing.demo.52north.org:8080/data/sf_only_roads-aoi.osm"
            },
            "_id": "INPUT2"
            }
        ],
        "output":[{
            "_mimeType": "application/x-zipped-shp",
            "_id": "CONFLATION_OUTPUT",
            "_transmission": "reference"
        },{
            "_mimeType": "text/plain",
            "_id": "CONFLATION_REPORT",
            "_transmission": "reference"
        }],
        "_service": "WPS",
        "_version": "2.0.0"
    }
}
```

The direct response to a asynchronously executed process is HTTP status code 201 (created) and the URL to obtain status information and finally the result. The URL will be returned in a HTTP header named *Location*. For synchronous execution, the result document will be returned after the process has finished.

Example of HTTP GET request for retrieving status information about a asynchonously executed process (NOTE: Request has been line-wrapped for easier reading).

```
http://tb12.dev.52north.org/wps-rest/processes/
testbed12.fo.DouglasPeuckerAlgorithm/jobs/
c731d14b-1de6-499c-9317-20224e056012
```

Example of WPS StatusInfo response encoded as JSON. The process is still running.

```
{
  "StatusInfo": {
    "JobID": "c731d14b-1de6-499c-9317-20224e056012",
    "Status": "Running",
    "Progress": 0
  }
}
```

After the process has finished, the progress element will be replaced by the URL to obtain the outputs

Example of WPS StatusInfo response encoded as JSON. The process has finished.

```
{
  "StatusInfo": {
    "JobID": "c731d14b-1de6-499c-9317-20224e056012",
    "Status": "Succeeded",
    "Output": "http://tb12.dev.52north.org/wps-
rest/processes/testbed12.lsa.HootenannyConflation/jobs/c731d14b-1de6-499c-9317-
20224e056012/outputs"
  }
}
```

Example of WPS Result response encoded as JSON.

```
{
  "Result": {
    "JobID": "c731d14b-1de6-499c-9317-20224e056012",
    "Output": [
      {
        "ID": "CONFLATION_OUTPUT",
        "Reference": {
          "_mimeType": "application/x-zipped-shp",
          "_href":
"http://tb12.dev.52north.org:80/wps/RetrieveResultServlet?id=c731d14b-1de6-499c-9317-
20224e056012CONFLATION_OUTPUT.b1172b1c-c9aa-495a-aa8b-62220ac93605"
        }
      },
      {
        "ID": "CONFLATION_REPORT",
        "Reference": {
          "_mimeType": "text/plain",
          "_href":
"http://tb12.dev.52north.org:80/wps/RetrieveResultServlet?id=c731d14b-1de6-499c-9317-
20224e056012****CONFLATION_REPORT.220391a6-4357-44e2-b5f3-c0a0983cedae"
        }
      }
    ]
  }
}
```

Lessons Learned

The WPS REST API simplifies the interaction between servers and clients. The HTTP methods with its clear semantics ease to understand the interface and its communication pattern. The usage of hypermedia encoded as JSON allows clients to browse from the Capabilities to the processes available, to navigate from these processes to jobs (running instances of these processes) and, once a job is finished, to browse to the resulting outputs of a job.

One difficulty encountered is the description of the API using the Capabilities document. In general, in order to interact with an HTTP based RESTful API, clients need to know

- which resources are offered and
- which HTTP methods need to be applied in order to retrieve or manipulate these resources

While the information about the resources offered can be easily embedded in the Contents section of the Capabilities document, the integration of the description about which HTTP methods are applicable on which resources is not straightforward. Usually, the OperationsMetadata section would be used for describing the operations offered, the endpoints and the operation parameter. However, since the OperationsMetadata have been defined for Web Services in Service-oriented Architectures, the description of REST APIs is not straightforward for the following reasons:

Only HTTP GET and POST are supported.

- There is no way to describe on which resources the HTTP methods, e.g. HTTP GET, can be applied.

For this reason, we omitted the OperationsMetadata section and described the application of the HTTP methods in the specification draft. In future, it would be desirable to also offer a machine-readable description of the API relying upon existing description language, e.g. the OpenAPI specification [OpenAPI_2016].

4.5.4. JSON and WMS

A WMS 1.3 server supporting GeoJSON as a GetFeatureInfo output format was deployed for Testbed 12 here:

http://tb12.cubewerx.com/a040/cubeserv?SERVICE=WMS&VERSION=1.3.0&REQUEST=GetCapabilities

The following example retrieves GeoJSON output is response to a GetFeatureInfo request:

Example of HTTP GET request for retrieving a feature info resource (NOTE: Request has been line-wrapped for easier reading).

```
http://tb12.cubewerx.com/a040/cubeserv?SERVICE=WMS&
      VERSION=1.3.0&
      LANGUAGE=en-CA,en&
      REQUEST=GetFeatureInfo&
      CRS=EPSG%3A3857&
      BBOX=-13641185.826,4546529.053,-13601935.412,4575116.502&
      WIDTH=1027&HEIGHT=748&
      LAYERS=USGS.Struct_Point&
      FORMAT=image%2Fx-jpegorpng&
      QUERY_LAYERS=USGS.Struct_Point&
      FEATURE_COUNT=10&
      I=812&
      J=345&
      INFO_FORMAT=application%2Fjson
```

The following GeoJSON document is retrieved by this URL:

Example of a feature info resource encoded as GeoJSON.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "fsetName": "Struct_Point",
      "geometry": {
        "type": "Point",
        "coordinates": [
```

```
          -122.2622829998311,
          37.87610699954206
        ]
      },
      "properties": {
        "PERMANENT_": "{F154F04D-FD02-4D8E-9B8E-E122331C931C}",
        "SOURCE_FEA": "115214",
        "SOURCE_DAT": "{B04D081F-2258-4584-B950-3659EE6688B8}",
        "SOURCE_D_1": "Schools ORNL nationwide schools dataset USGS provisional load",
        "SOURCE_ORI": "Oak Ridge National Laboratory, Geographic Information Sciences
and Technology Group",
        "DATA_SECUR": 5,
        "DISTRIBUTI": "E3",
        "LOADDATE": "2011-12-20 00:00:00.000 +00:00",
        "FTYPE": 730,
        "FCODE": 73006,
        "NAME": "Graduate Theological Union",
        "ISLANDMARK": 2,
        "POINTLOCAT": 1,
        "ADMINTYPE": null,
        "ADDRESSBUI": null,
        "ADDRESS": "2400 Ridge Rd",
        "CITY": "Berkeley",
        "STATE": "CA",
        "ZIPCODE": "94709-",
        "GNIS_ID": null,
        "FOOT_ID": null,
        "COMPLEX_ID": null
      }
    },
    {
      "type": "Feature",
      "fsetName": "Struct_Point",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -122.2614299998325,
          37.87587699954241
        ]
      },
      "properties": {
        "PERMANENT_": "{D8BFD704-2104-4EC2-A725-6CB66A6E7B97}",
        "SOURCE_FEA": "112127",
        "SOURCE_DAT": "{B04D081F-2258-4584-B950-3659EE6688B8}",
        "SOURCE_D_1": "Schools ORNL nationwide schools dataset USGS provisional load",
        "SOURCE_ORI": "Oak Ridge National Laboratory, Geographic Information Sciences
and Technology Group",
        "DATA_SECUR": 5,
        "DISTRIBUTI": "E3",
        "LOADDATE": "2011-12-20 00:00:00.000 +00:00",
        "FTYPE": 730,
```

```
        "FCODE": 73006,
        "NAME": "Church Divinity School of the Pacific",
        "ISLANDMARK": 2,
        "POINTLOCAT": 1,
        "ADMINTYPE": null,
        "ADDRESSBUI": null,
        "ADDRESS": "2451 Ridge Road",
        "CITY": "Berkeley",
        "STATE": "CA",
        "ZIPCODE": "94709-1211",
        "GNIS_ID": null,
        "FOOT_ID": null,
        "COMPLEX_ID": null
      }
    },
    ...
  ],
  "bbox": [
    -122.2624979998308,
    37.875506999543,
    -122.2614299998325,
    37.87680799954097
  ]
}
```

NOTE	Normally, packed JSON output is generated from by the A040 server but the output presented here has been wrapped and truncated to make it easier to read.

Lessons Learned

Providing a GeoJSON response for WMS GetFeatureInfo requests makes a lot of sense, since more and more client applications are web-browser based and JSON is the more natural representation in this environment. However, OGC 15-053 imposes certain complications that were questioned, and ultimately discarded, in the implementation of A040.

OGC 15-053 states that "we are not recommending returning the full geometry description because we assume that this is the work of a WFS GetFeature operation using identifiers recovered by the GetFeatureInfo request". The justification for suggesting that returning a geometry is out of scope for a WMS GetFeatureInfo response was not clear. The entire point of having a WMS GetFeatureInfo request is to:

1. give a WMS client the ability to query information about a feature without having to implement any WFS logic
2. give a WMS server the ability to provide information about a feature without having to implement a companion WFS

If a WMS client is interested in the geometry of a feature, it's likely for the purpose of plotting it on the map to highlight the feature(s) that the user has clicked on. Wouldn't it be convenient if the GeoJSON GetFeatureInfo response could include the geometry as part of the response? Of course,

GeoJSON can do exactly that so we saw no reason for needlessly complicating the situation by bringing a WFS into the picture.

Furthermore, returning a vector pointing from the user click location (which client already knows) to "a point that is in the interior or the border of the returned feature" is completely useless to the client. Then requiring the client to somehow determine the association to the appropriate WFS server and WFS feature type (see discussion below), formulate and issue an appropriate WFS GetFeature request, and be equipped to handle the WFS response, is a lot to ask of a simple WMS client especially in the case where the client simply wants to highlight a feature on the map.

The requirements imposed on a WMS server by this aspect of requirement 28 are also overly onerous. The requirement is basically suggesting that if a WMS wants to be able to serve geometries to its client — so that it can highlight them on a map for example — then the WMS must also include an implementation of a WFS server to handle that. This all seems very heavyweight for no good reason, especially since GeoJSON responses are naturally equipped to return the geometries right from the start.

If the concern is that returning geometries in a WMS GetFeatureInfo response uses unnecessary bandwidth for client applications that don't require the geometries, then our position is that using a bit of extra bandwidth like this is a small price to pay for the simplicity of the interface. The addition of two optional GetFeatureInfo parameters (Table 6) is proposed to streamline this:

Table 11. Additional proposed GetFeatureInfo request parameters

Parameter Name	Values	Description
GET_GEOMETRIES	Boolean	Indicates whether or not geometries should be returned in the GetFeatureInfo response. If FALSE, then (at least in the case of GeoJSON output) the geometry should be returned as null. If unspecified, it's at the server's discretion. The server is not obligated to honour this parameter.
SIMPLIFY_GEOMETRIES	Boolean	Indicates whether or not geometries returned in GetFeatureInfo response should be simplified to the resolution of the corresponding GetMap request. If unspecified, the default value is FALSE. The server is not obligated to honour this parameter.

The deployed TB12 implementation of the GetFeatureInfo operation includes these two parameters.

Finally, none of this is to say that returning the WFS feature identifier of the feature as part of the WMS GetFeatureInfo response is necessarily a bad thing. In fact, it might be useful to complex clients that really do need full WFS access to the feature (for example, to perform updates on it,

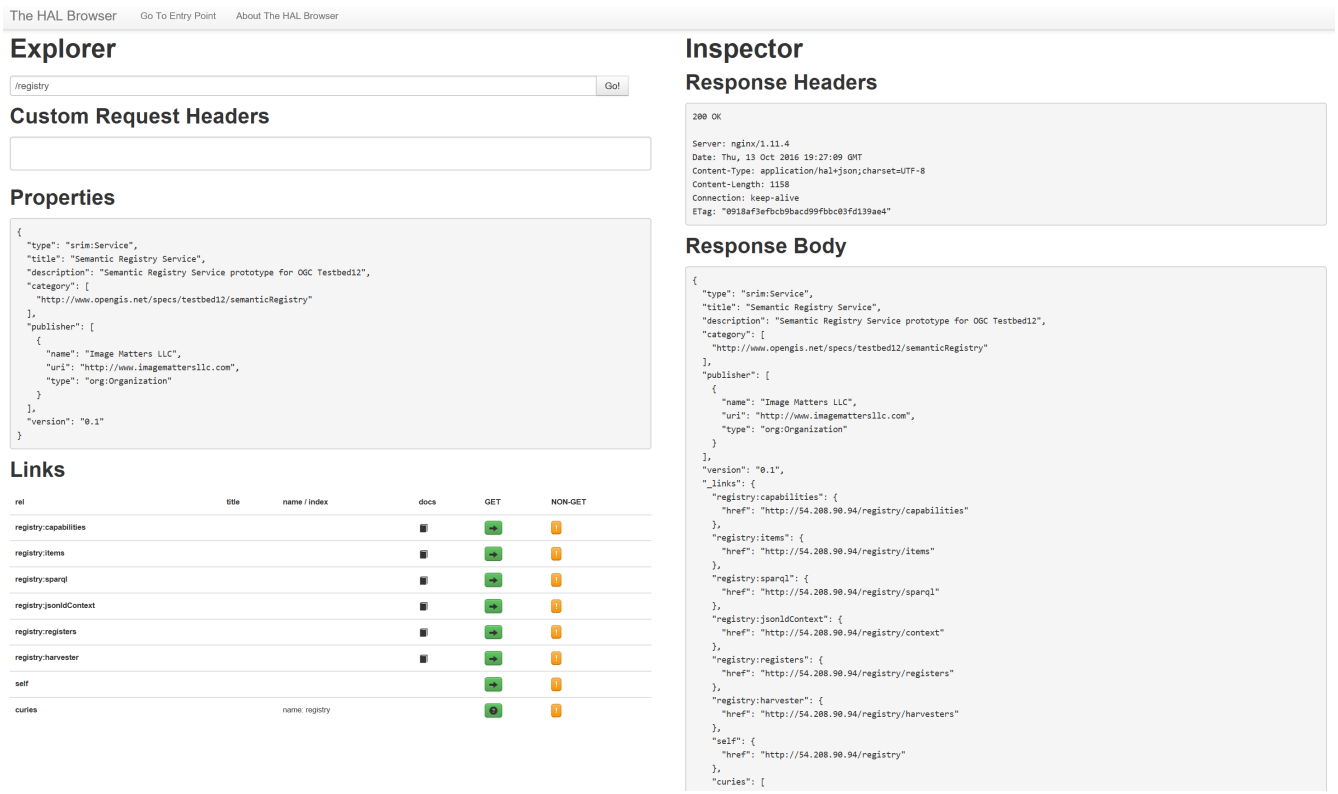
etc.). It's worth noting that a WFS feature identifier alone is not enough. The client application would have to also know:

1. The base URL of the association WFS server (it's not necessarily the same as the base URL of the WMS server being accessed)
2. The name of the associated WFS feature type (it's not necessarily the same as the name of the WMS layer being accessed)
3. The namespace of the associated WFS feature type (there is not way to deduce this from the WMS interface)

The mechanism(s) to allow the determination of these associations was certainly beyond the scope of OGC 15-053r1. However, OGC 16-043, "Testbed-12 Web Integration Service Engineering Report", proposes just such a mechanism in Clause 6, "GetAssociations operation".

4.5.5. JSON and Registry

Testbed 12 deployed an instance of the Image Matters Semantic Registry (previously named DCAT Rest Service). The service implements by default a Level 3 REST API (HAL+JSON). It also provides responses in Level 2 (JSON-LD) and as Linked Data (RDF, TTL, N-TRIPLES). HAL is a format based on JSON that establishes conventions for representing links and a breif examples of HAL+JSON in the HAL browser is shown below.



A sample response in JSON is shown below.

```
200 OK

Content-Type: application/hal+json;charset=UTF-8
Content-Length: 1158
```

```
ETag: "0918af3efbcb9bacd99fbbc03fd139ae4"


{
  "type": "srim:Service",
  "title": "Semantic Registry Service",
  "description": "Semantic Registry Service prototype for OGC Testbed12",
  "category": [
    "http://www.opengis.net/specs/testbed12/semanticRegistry"
  ],
  "publisher": [
    {
      "name": "Image Matters LLC",
      "uri": "http://www.imagemattersllc.com",
      "type": "org:Organization"
    }
  ],
  "version": "0.1",
  "_links": {
    "registry:capabilities": {
      "href": "http://54.208.90.94/registry/capabilities"
    },
    "registry:items": {
      "href": "http://54.208.90.94/registry/items"
    },
    "registry:sparql": {
      "href": "http://54.208.90.94/registry/sparql"
    },
    "registry:jsonldContext": {
      "href": "http://54.208.90.94/registry/context"
    },
    "registry:registers": {
      "href": "http://54.208.90.94/registry/registers"
    },
    "registry:harvester": {
      "href": "http://54.208.90.94/registry/harvesters"
    },
    "self": {
      "href": "http://54.208.90.94/registry"
    },
    "curies": [
      {
        "href": "http://www.opengis.net/rels/registry/{rel}",
        "name": "registry",
        "templated": true
      }
    ]
  }
}
```

4.6. API Management

API management is an important software engineering function for any organization trying to decide whether to use open geospatial REST to offer data and services. As the popularity of REST APIs has grown in the past few years, so too have the tools, best practices and consulting services that support organizations in API management. While these tools were initially developed for organizations to manage their organization-specific APIs, the tools can also be considered for use in API design in an open standards context. In partcicular, Testbed 12 participants have found the Open API Specification, formerly known as the Swagger specification, to be helpful.

Documenting APIs can be aided by using the approach of the OpenAPI Initiative (OAI). OAI is focused on defining a vendor neutral API Description Format based on the Open API Specification (OAS). The approach will allow specification of REST APIs using modular sub-elements. Subelements can then live on their own and be shared by multiple APIs.

Providing complete documentation of your API using OpenAPI is a W3C Data on the Web Best Practice. Best practice for APIs is being discussed in the OGC/W3C Spatial Data on the Web Working Group.

It is recommended that enterprises make use of automated API documentation where possible. These can often be synched directly with an implementation version, which helps to minimize divergence. Some also provide interactive (e.g. Swagger) documentation that allows inline requests to be made. This helps to lower the barrier of entry for developers and quickly builds understanding of the REST API.

Chapter 5. Summary

This document presented OGC Testbed 12 experience using open geospatial REST. For the purposes of this guide we limited REST to the Web (i.e. HTTP) - and then broke it down into four Levels. The Levels progress from Services to a focus on geospatial Resources, HTTP Verbs and linking Resources. These Levels of REST aren't discrete and exclusive – they build upon each other to help geospatial interoperability.

A key point in this guide is the recognition that open geospatial Resources include very complex information about the Earth – such as electro-optical, infrared, multispectral, hyperspectral and radar satellite measurements, sensor observations and much more. This means that as much as we want all Resources to fit neatly into Level 1, 2 or 3, the reality is some are better suited to certain URL structures or Levels.

Another important finding of Testbed 12 is that open geospatial REST allows applications to select the encoding format that is most useful to their mission. For example, Resources can be represented in different encodings such as XML or JSON, depending on whether applications need complex analytics or simpler representations like web display. That said, Testbed 12 assessed that traditional interfaces such Web Feature Service should no longer mandate XML, and allow deployment of GeoJSON-only profiles.

In summary, the technology integration experiments conducted in Testbed 12 showed that REST was able to represent a variety of real world phenomena as open geospatial Resources. The Resources can then be accessed and updated using the language of the World Wide Web - greatly simplifying development and deployment for geospatial enterprises.

Chapter 6. Revision History

Table 12. Revision History

Date	Release	Editor	Primary clauses modified	Descriptions
April 15, 2016	.1	Jeff Harrison	All	Initial Outline
June 30, 2016	.2	Jeff Harrison	All	First Draft
September 30, 2016	.31	Jeff Harrison	All	Second Draft
October 7, 2016	.35	Jeff Harrison	Updated examples from Testbed 12 ERs	Third Draft
October 13, 2016	.36	Jeff Harrison	Refined examples from Testbed 12	Fourth Draft
October 24, 2016	.37	Jeff Harrison	Edits from OGC IPTeam Review	Fifth Draft
October 31, 2016	.38	Jeff Harrison	Formatting and minor update to graphics	Sixth Draft
November 11, 2016	.39	Jeff Harrison	Edits from OGC Sponsor Review	Seventh Draft