Open Geospatial Consortium Inc.

Date:   2012-06-22

Reference number of this OGC® project document:    OGC 12-054r1

Version: 0.0.7

Category: OGC® Implementation Specification

Editor:   Clemens Portele, Satish Sankaran

GeoServices REST API — Part 1: Core

**Copyright notice**

Copyright © 2012 Open Geospatial Consortium, Inc. All Rights Reserved.
To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>.

**Warning**

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Document type:    OGC® Publicly Available Standard

Document subtype:    n/a

Document stage:    Draft

Document language:  English

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Draft copyright:

Copyright: Esri Inc.

The companies listed above have granted the Open Geospatial Consortium, Inc. (OGC) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version.

Contents

1. Scope 1

2. Conformance 1

3. References 4

4. Terms and Definitions 5

5. Conventions 5

5.1 Presentation of requirements and recommendations 5

5.2 Structure of resource descriptions 6

5.2.1 General remarks 6

5.2.2 Overview section 6

5.2.3 Resources 7

5.2.4 URI and HTTP request section 7

5.2.5 Resource / representation section 9

6. Fundamentals of the GeoServices REST API 9

6.1 Overview 9

6.2 REST and pragmatic considerations 10

6.2.1 General remarks 10

6.2.2 HTTP methods 11

6.3 Resources 12

6.3.1 General remarks 12

6.3.2 Controller resources 12

6.3.3 Commonly-Used Resources 13

6.4 Response Format 14

6.5 Extensibility 14

7. Core 14

7.1 Overview 14

7.2 General requirements for extensibility 15

7.3 Exceptions 15

8. Support for JSONP 16

9. Geometry information 17

9.1 Overview 17

9.2 Spatial reference 17

9.3 Point 19

9.4 Multipoint 19

9.5 Polyline 19

9.6 Polygon 20

9.7 Envelope 20

9.8 Geometry Array 21

10. Features 21

10.1 Overview 21

10.2 Feature 21

10.3 Fields 23

10.4 Feature Types, Layers and Tables 24

10.5 Feature Set 24

10.6 Feature ID Set 26

10.7 Range Domain 27

10.8 Coded Value Domain 27

10.9 Inherited Domain 28

11. Symbols, Labels, Renderers 28

11.1 Overview 28

11.2 Color 28

11.3 Symbol Objects 28

11.3.1 Overview 28

11.3.2 Simple Marker Symbol 29

11.3.3 Simple Line Symbol 30

11.3.4 Simple Fill Symbol 30

11.3.5 Picture Marker Symbol 31

11.3.6 Picture Fill Symbol 32

11.3.7 Text Symbol 33

11.4 Label Objects 34

11.4.1 Overview 34

11.4.2 Label Class 35

11.4.3 Labeling Info 37

11.5 Renderer Objects 38

11.5.1 Overview 38

11.5.2 Simple Renderer 38

11.5.3 Unique Value Renderer 39

11.5.4 Class Breaks Renderer 40

A.1 Conformance class: core 42

A.1.1 Test: core/extensibility 42

A.1.2 Test: core/exceptions 42

A.2 Conformance class: jsonp 43

A.2.1 Test: core/exceptions 43

A.3 Conformance class: geometry 43

A.3.1 Test: geometry/sr 43

A.3.2 Test: geometry/point 44

A.3.3 Test: geometry/multipoint 44

A.3.4 Test: geometry/polyline 44

A.3.5 Test: geometry/polygon 44

A.3.6 Test: geometry/envelope 45

A.3.7 Test: geometry/array 45

A.4 Conformance class: feature 45

A.4.1 Test: feature/feature 45

A.4.2 Test: feature/layerOrTable 45

A.4.3 Test: feature/featureSet 46

A.4.4 Test: feature/featureIdSet 46

A.4.5 Test: feature/rangeDomain 46

A.4.6 Test: feature/codedValueDomain 46

A.4.7 Test: feature/inheritedDomain 47

A.5 Conformance class: symbol 47

A.5.1 Test: symbol/color 47

A.5.2 Test: symbol/sms 47

A.5.3 Test: symbol/sls 47

A.5.4 Test: symbol/sfs 47

A.5.5 Test: symbol/pms 48

A.5.6 Test: symbol/pfs 48

A.5.7 Test: symbol/ts 49

A.5.8 Test: symbol/label 49

A.5.9 Test: symbol/labelInfo 49

A.5.10 Test: symbol/simpleRenderer 49

A.5.11 Test: symbol/uniqueValueRenderer 50

A.5.12 Test: symbol/classBreaksRenderer 50

Preface

The “Esri GeoServices REST Specification Version 1.0” was originally developed by Esri to provide interoperability between ArcGIS Server and the broader information technology community. The Esri specification had been widely implemented by Esri users and business partners over 4 years. In 2010 it was released as a non-proprietary open specification and has been implemented by developers outside of the Esri user community.

In 2011, Esri has offered the GeoServices REST API for consideration to become an OGC standard. An OGC Standards Working Group was formed to document the specification in conformance with the modular specification policy of the OGC and to address comments received from the OGC membership and during the public review.

This candidate standard is designed to be implemented without the use of Esri products.

Submitting organizations

The following organizations submitted this Implementation Specification to the Open Geospatial Consortium Inc.:

Esri Inc.

interactive instruments GmbH

Oracle USA

52°North

Submission contact points

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

|  |  |
| --- | --- |
| CONTACT | COMPANY |
| 1. Satish Sankaran
 | 1. Esri Inc.
 |
| 1. Keith Ryden
 | 1. Esri Inc.
 |
| 1. Clemens Portele
 | 1. interactive instruments GmbH
 |
| 1. John Herring
 | 1. Oracle USA
 |
| 1. Andreas Wytzisk
 | 1. 52° North
 |

Revision history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Release | Author | Paragraph modified | Description |
| 1. 29 Aug 2011
 | 1. 0.0.1
 | 1. CP, SS
 | 1. All
 | 1. First internal draft
 |
| 1. 16 Oct 2011
 | 1. 0.0.2
 | 1. CP, SS
 | 1. All
 | 1. Revised based on discussion in Boulder, convetions clause added, ATS added, request/URI requirements formalised
 |
| 1. 25 Nov 2011
 | 1. 0.0.3
 | 1. CP, SS
 | 1. All, in particular 7, 8 and Annex A
 | 1. Removed catalog and clarified exception codes, added JSONP conf class
 |
| 1. 23 Jan 2012
 | 1. 0.0.4
 | 1. CP, SS
 | 1. 6.3.2, 10+
 |  |
| 1. 01 May 2012
 | 1. 0.0.5
 | 1. CP
 | 1. All
 | 1. Include changes identified in review process
2. Update versioning rules section based on 01 May telecon
 |
| 1. 31 May 2012
 | 1. 0.0.6
 | 1. CP
 |  | 1. RFC package for OAB review
 |
| 1. 22 Jun 2012
 | 1. 0.0.7
 | 1. CP
 |  | 1. Update URIs
 |

Changes to the OGC® Abstract Specification

The OGC**®** Abstract Specification does not require changes to accommodate this OGC**®** standard.

Versioning Rules

The GeoServices REST API series of standards provides a standard way for web clients to communicate with geographic information system (GIS) servers based on Representational State Transfer (REST) principles. It is based on existing, proven and widely deployed technology.

To build on the existing ecosystem of services and applications supporting the API, this standard has been specified so that it is backwards compatible with existing GeoServices REST API documents and implementations already in widespread use within the mass market. Backwards compatibility will remain essential in the future to continue to build on and increase the base of services and applications.

At the same time, this specification will be open to additional capabilities, typically captured in additional, optional conformance classes, whenever such extensions have been proven in implementation and adoption in the marketplace. Extensions will typically be either new GIS capabilities or support for new web technologies.

Future versions of this standard will be considered backwards compatible, if the request parameters and JSON representations of a resource are compatible with previous versions, i.e. the new version will only introduce the following changes:

* add new service types
* add new resources to an existing service type
* add new optional parameters to an existing resource without changing the default behaviour
* add new well-known values for an existing parameter that is not a JSON object
* add new optional properties to the JSON schema of parameters that are JSON objects
* add new optional properties to the JSON schema of a JSON resource representation
* add support for new HTTP methods
* add support for new formats

A candidate for an extension is the following: The standard deviates from a "pure" RESTful approach whenever this would have created barriers for implementers, for example, due to limited support for HTTP in existing web client frameworks. As a result, the HTTP methods PUT and DELETE are not used as part of the framework and the content types used in POST are limited to those supported by HTML. An increasing demand for services that support a larger part of the HTTP stack may, for example, be addressed by additional conformance classes that complement the existing operations for updating and deleting resources that use POST. The current conformance classes have already been organized with such additional conformance classes in mind.

One aspect to consider in adding additional conformance classes is to avoid too many parallel options for essentially the same capability in the interface. A new extension will only be considered for standardisation, if its implementability and adoption by application developers has been demonstrated.

While backwards compatibility is essential, both GIS and the web and its technologies are evolving. In addition to a strategy for extending the standard to include new capabilities as outlined above we also need a complementary strategy for deprecating parts of the API that are no longer used in practice.

In the Geoservices REST API, changes that are neither backwards compatible nor a bugfix will only be made in a major revision. Changes that remove a part of the standard, e.g. support for a controller resource, will only be made if the part has been deprecated for at least one year. Such deprecations will be widely announced possibly including an OGC press release.

While the above statements and backwards compatibility rules will be adhered to as much as possible, in certain rare situations it might be necessary to break compatibility without introducing a new major version. If such a change is required, it will be documented in a very visible way both in the new version and the revision notes.

All versions of the Geoservices REST API will remain available on the OGC website and continue to be maintained with corrigenda, if errors are identified.

Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights. However, to date, no such rights have been claimed or identified.

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 specification set forth in this document, and to provide supporting documentation.

This document is part 1 of the GeoServices REST API series:

Part 1: Core

Part 2: Catalog

Part 3: Map Service

Part 4: Feature Service

Part 5: Geometry Service

Part 6: Image Service

Part 7: Geoprocessing Service

Part 8: Geocoding Service

The relationship with other parts of the OGC standards baseline is described in document 12-062.

GeoServices REST API — Part 1: Core

# Scope

The GeoServices REST API provides a standard way for web clients to communicate with geographic information system (GIS) servers based on Representational State Transfer (REST) principles. Clients issue requests to the resources on the server identified by structured URLs. The server responds with map images, text-based geographic information, or other representations of resources that satisfy the request.

This document specifies commonly used resources in an implementation of the GeoServices REST API as well as general requirements.

# Conformance

Conformance with this standard shall be checked using all the relevant tests specified in Annex A (normative) of this document. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site[[1]](#footnote-1).

This Standard establishes five requirements classes and corresponding conformance classes - the abstract core conformance class of the GeoServices REST API, a conformance class for JSONP support and a conformance class for JSON representations of geometry objects. All other parts of the GeoServices REST API series specify extensions to this conformance class.

In order to conform to this OGC™interface standard, a software implementation shall choose to implement:

* The conformance class "core".
* At least one conformance class of standardization target type "web service" in other parts of the GeoServices REST API series.
* EXAMPLE An implementation may implement the core conformance class and the mapservice conformance class specified in the standard "GeoServices REST API – Map Service".

NOTE The geometry, feature and symbol conformance classes are referenced from extensions.

All requirements-classes and conformance-classes described in this document are owned by the standard identified as http://www.opengis.net/spec**/gsr/1.0**. Requirements and conformance test URIs defined in this document are relative to this URI.

Any implementation claiming conformance with a conformance class shall pass all the tests in the associated abstract test suite. Table 1 summarizes the requirements and conformance tests associated per conformance class.

Table 1 – Conformance class summary

|  |  |  |
| --- | --- | --- |
| **core** | **Title** | GeoServices REST API Core |
| **Standardization target type** | Web service |
| **Dependencies** | **http://tools.ietf.org/html/rfc3986****http://tools.ietf.org/html/rfc2616****http://tools.ietf.org/html/draft-gregorio-uritemplate-05** **http://tools.ietf.org/html/rfc4627****http://tools.ietf.org/html/draft-zyp-json-schema-03** |
| **Requirements** | All requirements in Clause 7 |
| **Conformance tests** | Annex A.1 |
| **jsonp** | **Title** | JSONP support |
| **Standardization target type** | Web service |
| **Dependencies** | **conf/core** |
| **Requirements** | All requirements in Clause 8 |
| **Conformance tests** | Annex A.2 |
| **geometry** | **Title** | Geometries |
| **Standardization target type** | JSON representation |
| **Dependencies** | **http://tools.ietf.org/html/rfc4627****http://tools.ietf.org/html/draft-zyp-json-schema-03****http://www.opengis.net/doc/sfa/1.1/clause/6.4** |
| **Requirements** | All requirements in Clause 9 |
| **Conformance tests** | Annex A.3 |
| **feature** | **Title** | Features |
| **Standardization target type** | JSON representation |
| **Dependencies** | **conf/geometry** |
| **Requirements** | All requirements in Clause 10 |
| **Conformance tests** | Annex A.4 |
| **symbol** | **Title** | Symbols, Labels and Renderer |
| **Standardization target type** | JSON representation |
| **Dependencies** | **http://tools.ietf.org/html/rfc4627****http://tools.ietf.org/html/draft-zyp-json-schema-03** |
| **Requirements** | All requirements in Clause 11 |
| **Conformance tests** | Annex A.5 |

The following figure provides an overview of the hierarchy of conformance classes. The resources shown in grey are examples specified in other parts of the GeoServices REST API series.

Figure 1 – Hierarchy of conformance classes

# References

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

Uniform Resource Identifier (URI): Generic Syntax, IETF RFC 3986 (2005), http://tools.ietf.org/html/rfc3986

Hypertext Transfer Protocol -- HTTP/1.1, IETF RFC 2616 (1999), http://tools.ietf.org/html/rfc2616

URI Template, IETF draft 05 (2011), http://tools.ietf.org/html/draft-gregorio-uritemplate-05

The application/json Media Type for JavaScript Object Notation (JSON), IETF RFC 4627 (2006), http://tools.ietf.org/html/rfc4627

A JSON Media Type for Describing the Structure and Meaning of JSON Documents, IETF draft 03 (2010), http://tools.ietf.org/html/draft-zyp-json-schema-03

OGC Simple Feature Access *–* Part 1: Common Architecture, Version 1.1, OGC document 05-126

NOTE Same as ISO 19125-1:2003, *Geographic information — Simple feature acess — Part 1: Common architecture*

# Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r9], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

# Conventions

## Presentation of requirements and recommendations

Requirements are presented using the following style:

|  |
| --- |
| **Req <number>** <requirement text><conformance class id>/<local requirement id> |

<number> is a unique number within the document.

<requirement text> is the requirement itself. Normative verbs like SHALL are written in capitals.

The smaller blue text at the bottom of the box <conformance class id>/<local requirement id> is the path fragement that appended to the URI identified in Clause 2 of the document (for example, http://www.opengis.net/spec/gsr/1.0 for this document) provides the URI of the requirement which can be used to unamibiguously identify the requirement.

Recommentdation are written in bold text, normative verbs like SHOULD are written in captials.

## Structure of resource descriptions

### General remarks

All clauses in this standard and all other parts of GeoServices REST API series that specify a conformance class for one or more resources follow the same structure.

EXAMPLE See Clause 7 of this document.

### Overview section

The first subclause provides an overview using descriptive text as well as an overview table with three columns:

* "Resource": The name of the resource on which the conformance class puts requirements.
* "Parameters": For the resource identified in the first column the parameters are listed for which the conformance requires support.

Where specific values for a parameter have to be supported, these are identified in the column, too. Example: "f=json".

* "Resource representation": This column provides an overview of the requirements related to representations of the resource returned from a request to a resource including which parts of the JSON Schema describing with the JSON representation of the resource are related to the conformance class.

EXAMPLE The following table contains a subset of the overview table of the Map Service Core conformance class.

Table 2 – Example overview table

|  |  |  |
| --- | --- | --- |
| **Resource** | **Parameters** | **Resource representation** |
| Map Service Root | f=json | JSON representation validJSON schema elements supported:- serviceDescription- mapName... |
| Map | f=jsonf=imagebbox... | JSON representation validAll JSON schema elements supportedImage has correct format and size |

### Resources

Following the overview, a subclause is provided for each resource listed in the overview. Again, each subclause follows a common structure and consists of two sections, the first specifying the requirements related to the URI of the resource and to the HTTP requests and the second specifying the requirements related to representations of the resource returned in the response. These are described in the following two subclauses.

In cases where a dependency exists on another conformance class that already specifies requirements on the resource and which is extended by this conformance class, the dependency is specified explicitly in a third section.

### URI and HTTP request section

The section on the URI and HTTP request typically contains two requirements which reference two tables. In addition one or more examples are typically provided.

The first requirement provides specifics about the URI of the resource and the HTTP method(s) that may be used. The second requirement provides specifics about the parameters (if any) that have to be supported by an implementation. The requirements look like this:

|  |
| --- |
| **Req <n1>** The <name> resource SHALL accept requests that conform to the URI template in Table <t1> and use any HTTP method identified in the same table.<conformance class id>/request |

|  |
| --- |
| **Req <n2>** A <name> resource SHALL support all parameters and values specified in Table <t2>.<conformance class id>/parameters |

<name> is the name of the resource type.

The table <t1> is named as "<name> reference" and contains the following information, each in a separate row:

* "URI template": The applicable URI template (using the IETF draft specification referenced in Clause 3).
* "HTTP methods": The HTTP methods that must be supported by a compliant implementation (GET and/or POST[[2]](#footnote-2)). For POST also the MIME type of the payload is specified.
* "Parent Resource Type": The resource type of a resource that is the parent resource and that this resource depends upon. Also, the resource is usually reached navigating from the parent resource.
* "Child Resource Types": Resource types of resources to which this resource is a parent resource. If a resource has no child resources the respective row is usually omitted.

EXAMPLE 1 The following table contains the table of the Export Map resource.

Table 3 – Example reference table

|  |  |
| --- | --- |
| **URI template** | {+mapServiceURI}/export{?f,bbox,size,dpi,format,layers,transparent} |
| **HTTP methods** | GETPOST (application/x-www-form-urlencoded) |
| **Parent Resource Type** | Map Service Root |

The table <t2> is named as "<name> parameters" and contains one row per parameter. For each parameter the following information is provided:

* A textual description of the parameter.
* "Required": Identifies whether the parameter has to be provided with the request ("Yes"). If the parameter is not required ("No") the default value/behaviour is specified.
* "Syntax": The syntax of parameter values that have to be supported by compliant implementations. Augmented BNF for Syntax Specifications (ABNF, IETF RFC 5234) is used to specify the syntax. The basic rules used in parameter tables are specified in Annex B.

Extensions of the conformance class may restrict or extend the syntax used for a parameter, but may not change requirements related to the semantics of parameter values specified in the conformance class.

* "Example": An example parameter name/value pair.

If a resource has no parent or child resources the respective rows may be omitted.

EXAMPLE 2 The following table contains a subset of the table of the Map resource.

Table 4 – Example parameters table

|  |  |
| --- | --- |
| **Parameter** | **Details** |
| f | The response format.  |
| Required | Yes |
| Syntax | "json" / "image" |
| Example | f=json |
| size | The size (width \* height) of the exported image in pixels. |
| Required | No. Default: "400,400" |
| Syntax | WIDTH "," HEIGHT |
| Example | size=600,550 |

### Resource / representation section

The section on the resources and their representations typically contains one or more requirements.

As the standard representation for resources in the GeoServices REST API is JSON, for most resources a JSON Schema (using the IETF draft specification referenced in Clause 3) is specified and a requirement concerns the validty against this schema. I.e., a typical requirement is:

|  |
| --- |
| **Req <n>** The JSON representation of a [response to a request on a] <name> resource SHALL validate against the JSON Schema <schema uri in OGC schema repository> or in case of an exception against JSON Schema http://schemas.opengis.net/gsr/1.0/exception.json. <conformance class id>/valid |

Additional requirements are provided, if the specific constraint cannot be expressed in JSON Schema.

In addition one or more examples of a JSON representation of the resource are typically provided.

# Fundamentals of the GeoServices REST API

## Overview

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

This document specifies the catalogue, the other services are specified in additional parts of this series of standards.

## REST and pragmatic considerations

### General remarks

This standard is based on both RESTful principles and pragmatic considerations that were and are driven by support for various aspects of the HTTP protocol in commonly used environments like JavaScript, Adobe Flex or Microsoft Silverlight, in commonly used web browsers, or in proxys. These typically do not offer complete support for all of the HTTP standard.

Examples for issues:

* Lacking web browser support for HTTP PUT and DELETE
* Some rich internet application clients do not fully support PUT and DELETE (see also the next sub-clause)
* From a practical implementation standpoint from within the browser for HTML applications one may want to use HTTP GETs via dynamic script tags whenever this can be done safely. Dynamic script tags for GET operations are cacheable and do not force clients on different domains to go through a proxy.
* HTTP POST is needed whenever the size of the URL may be longer than 2048 characters. This isn’t usually a factor for most API designs, but in the context of geographic information it happens quite frequently (serialized geometries can easily be bigger than 2000 characters).
* Often firewalls and proxies strip out HTTP PUTs and DELETEs. This can be mitigated by forcing SSL for all requests, but this is not practical. Some RESTful APIs recommend HTTP method overloading to get around this, which would be a hack. In this case one would use POST, but in the header (or in the query parameter) one specifies that one really wants to do a PUT or DELETE.
* Cross domain scripting needs have been mostly solved using proxy servers and support for JSONP (JSON with padding[[3]](#footnote-3), see Clause 8). These make it impossible to support all the HTTP error codes. As discussed in Clause 8, most responses with JSON content will use an HTTP status code 200 and the JSON content will either be a resource representation, the result of a controller resource operation or an exception.
* MIME types are advertised using query parameters in the URL (e.g., "?f=json") rather than using the HTTP headers. This too can be attributed to the fact that many times proxy servers tend to strip out header information and hence a more practical/safer approach of using parameters in the URL has been used for this purpose.

The GeoServices REST API is stateless and each request contains all the information necessary for successful processing.

The book "RESTful Web Services Cookbook" from Subbu Allamaraju describes some practical recipes for RESTful web services and in general the patterns used in the GeoServices REST API can be found in the cookbook, e.g. see recipes 2.6, 8.1 and 8.3.

### HTTP methods

Since the API was originally developed ago, the situation with respect to support for other HTTP methods beside GET and POST has improved in browsers and client frameworks. On the other hand, users may still be using technology that is not cutting edge and a conservative approach to using capabilities supported only in newer systems seems expedient.

The API uses GET and POST only. GET is supported for all access that is safe and idempotent. Typically also POST with URL-encoded payload is supported which may be necessary in case of long parameter values.

For all changes to the server (creating new resources or changing/deleting them), POST is used, typically also URL-encoding the parameters in the payload. The only deviation is when multipart content is transferred, e.g. when adding an attachment.

I.e., for the communication between the client and the server, only those technologies are used that are supported in HTML forms as all web browsers and frameworks can be expected to support this[[4]](#footnote-4).

NOTE It is worth to note that intermediate drafts of HTML5 had added support for PUT and DELETE, but removed this in later versions. However, discussions seem to be ongoing[[5]](#footnote-5).

By now, support for PUT and DELETE outside of HTML is less of an issue, but still using GET and POST seems to be much easier in commonly-used client frameworks like Microsoft Silverlight and Adobe Flex.

* Microsoft Silverlight: The current version includes support for PUT and DELETE. However, the standard (and simpler) WebClient class continues to support GET and POST only, but a new HttpWebRequest class has been added to support PUT and DELETE, too[[6]](#footnote-6).
* Adobe Flex: The situation with Flex seems to be similar, but even more restricted. The standard methods supported for accessing HTTP services are GET and POST. HTTP PUT and DELETE only seem to be supported in desktop applications running in Adobe AIR or with Adobe LiveCycle Data Services or BlazeDS which also supports additional HTTP methods[[7]](#footnote-7).

As one of the goals of this standard is to reach a large developer base and enable them to use the tools that they are familiar with, restricting the API to GET and POST still seems appropriate[[8]](#footnote-8).

## Resources

### General remarks

The GeoServices REST API works with a hierarchy of resources and returns representations of resources to the client.

Each resource has a unique URL. Resources MAY support parameters.

### Controller resources

Some resources are controller resources (also called "operations") that either

* edit information in the server or
* query the information in the server and create resources that are not persistently stored on the server and that are not made available with their own URI, but returned in the response from the controller resource.

Controller resources of the second type could also be viewed simply as accessing existing resources on the server, while in general these will be dynamically created by the controller resource. In most cases these resources will support both GET and POST. The use of GET is usually preferable as the responses to these requests may be cached.

The use of verbs in the names of controllers (e.g. "addAttachement") differs from the typical current practice in RESTful APIs. In the case of the Feature Service controller resources addAttachement, updateAttachement and deleteAttachement (similar for feature editing) this is a result of using POST instead of PUT or DELETE. If the HTTP methods would be the verbs then POST, PUT, DELETE could be invoked upon the attachement aggregate (POST) or the attachement itself (PUT, DELETE). However, since controllers are used that all use POST only the different operations (adding, updating, deleting) have to be distinguished and thus the verbs become part of the URI.

In other cases (e.g. submitJob or executeTask) there is no apparent reason for using verbs except to make the URI as self-descriptive as possible - again since only a limited number of HTTP methods are used in the API.

### Commonly-Used Resources

This document specifies a number of resources and their JSON representation. These resources used by multiple services of the GeoServices REST API.

The following figure provides an overview.

Figure 2 – Resource overview

## Response Format

The URL of many resources in the GeoServices REST API has a parameter, “f”, that denotes the representation / response format. Developers may support resource representations in various data formats in addition to JSON, including, for example, HTML and KMZ.

This standard requires that at least the JSON response format is implemented for most resources, and the mandatory schema for the JSON representation of and examples for each resource type are provided.

For resources which support the parameter "f", the JSON representation of the resource is retrieved by appending the parameter value "f=json" to the URI of the resource.

Support for other formats is optional, and they can be accessed through the f parameter; however, representations other than JSON are in general not detailed in this standard.

## Extensibility

The GeoServices REST API is designed to be extended. Extensions may be implementation specific or they may be standardized.

The requirements of the Geoservices REST API standards are phrased with this design goal in mind. In particular, the following types of extensions are intended to be possible in dependent conformance classes.

* Additional resources
* Additional parameters in requests
* Extensions to the parameter value syntax of a parameter (as long as the semantics of the parameter values in the base conformance classes are unchanged)
	+ An example would be additional representations / response formats, i.e. additional values to the f parameter, or the support of additional geometry types
* Additional optional properties in resource representations
	+ JSON representations of a resource may include properties not included in the standardised schemas – this is permitted by JSON Schema

# Core

## Overview

The core conformance class is abstract and specifies principles that apply to all services of the GeoServices REST API. These are:

* Safety and idempotency of HTTP GET requests
* Requirements for extensibility
* Requirements for exceptions

## Safety and idempotency of HTTP GET requests

|  |
| --- |
| * + 1. If a request uses the HTTP GET method, the request SHALL be safe and idempotent.

core/get |

NOTE This behaviour is recommended by HTTP (RFC 2616).

## General requirements for extensibility

|  |
| --- |
| * + 1. If a request contains a parameter that is not specified in a requirements class implemented by the implementation the service SHALL ignore the unknown parameter.

core/unknownParameter |

|  |
| --- |
| * + 1. If a request contains a value of a parameter that is not specified in a requirements class implemented by the implementation the service SHALL raise an exception.

core/unknownParameterValue |

NOTE This standard does not specify requirements on clients of the GeoServices REST API, but it is good practice that clients ignore unknown elements in the reponse.

## Exceptions

|  |
| --- |
| * + 1. A request to a resource of a Geoservice REST API implementation SHALL result in an exception, if and only if the request violates a request requirement, unless an internal processing exception occurs.

core/exception |

|  |
| --- |
| * + 1. The JSON representation of an exception SHALL validate against the JSON Schema http://schemas.opengis.net/gsr/1.0/exception.json.

core/exceptionValid |

**Example**

An exception returned by a request to a Geometry service with invalid coordinate transformation parameters, e.g.

http://mysrv/rest/services/Geometry/GeometryServer/project?

inSR=a&

outSR=a&

geometries=aa&

f=json

{

 "error" :

 {

 "code" : 400,

 "message" : "Unable to complete Project operation.",

 "details" : [

 "'inSR' parameter is invalid",

 "'outSR' parameter is invalid",

 "The specified geometry is not in the correct format."

 ]

 }

}

NOTE The error messages are not standardized.

|  |
| --- |
| * + 1. The error code SHALL be a HTTP 1.1 status code (see RFC 2616, 6.1.1) that fits the error reported in the "message" and "details" properties.

core/exceptionCode |

# Support for JSONP

JSONP ("JSON with padding") is a convenient and popular method to overcome cross domain restrictions. Cross domain resources are loaded through script injection.

|  |
| --- |
| * + 1. If a request includes the parameter "f" with the value "json" and a parameter "callback", the response SHALL be "<callbackValue>(<originalJSONResponse>);" where <callbackValue> is the value of the parameter "callback" in the request and <originalJSONResponse> the result of the request, if the callback parameter had been omitted.

jsonp/callback |

I.e., the client may make the request with a special *callback* parameter and the server responds with a JSON padded response that looks like:

 callbackValue(*{original\_json\_response}*);

NOTE One of the inconveniences of using JSONP is that the server always needs to respond with an HTTP status of 200. Otherwise the browser’s network stack rejects the response from the server, and the client callbacks are never even called. To overcome this, the GeoServices REST API responds with errors wrapped inside the response (see the next sub-clause).

# Geometry information

## Overview

JSON representations of geometry objects (points, multi points, polylines, polygons) and arrays of geometry objects, envelopes and references to coordinate reference systems are used in various parts of the GeoServices REST API.

## Spatial reference

Features on a map refer to the actual locations of the objects they represent in the real world. The positions of objects on the earth's spherical surface are measured in degrees of latitude and longitude, also known as geodetic coordinates. Though latitude and longitude can locate exact positions on the surface of the earth, they are not uniform units of measure; only along the equator does the distance represented by one degree of longitude approximate the distance represented by one degree of latitude.

To overcome measurement difficulties, data is often transformed from the geodetic coordinate reference system on the ellipsoid to the planar surface in a projected coordinate reference system. Projected coordinate reference systems describe the distance from an origin (0,0) along two separate axes: a horizontal x-axis and a vertical y-axis.

Because the earth is round and maps are flat, getting information from the curved surface to a flat one involves a mathematical formula called a map projection. A map projection transforms longitude and latitude to x, y coordinates in a projected coordinate system.

The term *coordinate reference system,* which includes both geodetic and projected coordinate reference systems, is used to describe the information about the projection, as well as other specifics such as datum, units, and meridians.

Coordinate reference systems are specified by well-known text strings (WKT).

For well-known coordinate reference systems a well-known identifier (WKID) may be used. For these coordinate reference systems their WKT is provided.

|  |
| --- |
| * + 1. The JSON representation of a spatial reference SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/spatialreference.json**.

geometry/validSR |

**Example 1: "wkid"**

{ "wkid" : 4326 }

**Example 2: "wkt" of a geodetic coordinate reference system**

{ "wkt" : "GEOGCS[\"GCS\_WGS\_1984\",DATUM[\"D\_WGS\_1984\",SPHEROID[\"WGS\_1984\

",6378137,298.257223563]],PRIMEM[\"Greenwich\",0],UNIT[\"Degree\",

0.017453292519943295]]"}

**Example 3: "wkt" of a projected coordinate reference system**

{ "wkt" : "PROJCS[\"WGS\_1984\_Web\_Mercator\_Auxiliary\_Sphere\",GEOGCS[\"GCS\_WGS\_ 1984\",DATUM[\"D\_WGS\_1984\",SPHEROID[\"WGS\_1984\",6378137.0,298.257223563]],PRIMEM[\"Greenwich\",0.0],UNIT[\"Degree\",0.0174532925199433]],PROJECTION[\"Mercator\_Auxiliary\_Sphere\"],PARAMETER[\"False\_Easting\",0.0],PARAMETER[\"False\_Northing\",0.0],PARAMETER[\"Central\_Meridian\",0.0],PARAMETER[\"Standard\_Parallel\_1\",0.0],PARAMETER[\"Auxiliary\_Sphere\_Type\",0.0],UNIT[\"Meter\",1.0]]"}

|  |
| --- |
| * + 1. Each spatial reference SHALL contain one property. The property SHALL be "wkid" or "wkt".

geometry/wkidXorWkt |

|  |
| --- |
| * + 1. If the property is "wkid", the value SHALL be one of the entries in the document <http://schemas.opengis.net/gsr/1.0/sr.json>. The coordinate reference system is specified by the "wkt" property of the relevant JSON object in that document.

geometry/wkid |

**EXAMPLE** In the document http://schemas.opengis.net/gsr/1.0/sr.json, the entry for "4326" and "102100" is (extending the spatial reference schema):

{ "wkid" : 4326,

 "wkt" : "GEOGCS[\"GCS\_WGS\_1984\",DATUM[\"D\_WGS\_1984\",SPHEROID[\"WGS\_1984\

",6378137,298.257223563]],PRIMEM[\"Greenwich\",0],UNIT[\"Degree\",

0.017453292519943295]]",

 "uri" : "http://www.opengis.net/def/crs/gsr/4326" }

{ "wkid" : 102100,

 "wkt" : "PROJCS[\"WGS\_1984\_Web\_Mercator\_Auxiliary\_Sphere\",GEOGCS[\"GCS\_WGS\_ 1984\",DATUM[\"D\_WGS\_1984\",SPHEROID[\"WGS\_1984\",6378137.0,298.257223563]],PRIMEM[\"Greenwich\",0.0],UNIT[\"Degree\",0.0174532925199433]],PROJECTION[\"Mercator\_Auxiliary\_Sphere\"],PARAMETER[\"False\_Easting\",0.0],PARAMETER[\"False\_Northing\",0.0],PARAMETER[\"Central\_Meridian\",0.0],PARAMETER[\"Standard\_Parallel\_1\",0.0],PARAMETER[\"Auxiliary\_Sphere\_Type\",0.0],UNIT[\"Meter\",1.0]]",

 "uri" : "http://www.opengis.net/def/crs/gsr/102100" }

|  |
| --- |
| * + 1. Every value of the property "wkt" SHALL conform to the requirements in OGC Simple Feature Access – Part 1: Common Architecture, sub-clause 6.4.

geometry/wkt |

In the Geoservices REST API all geodetic coordinate reference systems use the axis order longitude, latitude and all projected coordinate reference systems use the axis order easting, northing.

NOTE The names “easting” and “northing” are mainly conventions and do not necessarily reflect the direction of the axes. ISO 19111:2007 / Topic 2, sub-clause 9.2.2 and Table 16 put a constraint to use northing (or southing) and easting (or westing) as the axis names for projected CRS. I.e., these axis names are also used, for example, for polar stereographic projections at the south pole where both axes are actually facing north (at 0°E and 90°E).

## Point

A point contains x and y fields along with a spatialReference field.

|  |
| --- |
| * + 1. The JSON representation of a point SHALL validate against the JSON Schema http://schemas.opengis.net/gsr/1.0/point.json.

geometry/validPoint |

**Example:**

{ "x" : -118.15, "y" : 33.80, "spatialReference" : {"wkid" : 4326} }

## Multipoint

A multipoint contains an array of points and a spatial reference. Each point is represented as a two-element array. The 0 index is the x-coordinate and the 1 index is the y-coordinate.

|  |
| --- |
| * + 1. The JSON representation of a multipoint SHALL validate against the JSON Schema http://schemas.opengis.net/gsr/1.0/multipoint.json.

geometry/validMultipoint |

**Example:**

{

"points" : [ [-97.06138,32.837], [-97.06133,32.836], [-

97.06124,32.834], [-97.06127,32.832] ],

"spatialReference" : {"wkid" : 4326}

}

## Polyline

A polyline contains an array of paths and a spatial reference. Each path is represented as an array of points. Each point in the path is represented as a two-element array. The 0 index is the x-coordinate and the 1 index is the y-coordinate.

|  |
| --- |
| * + 1. The JSON representation of a polyline SHALL validate against the JSON Schema http://schemas.opengis.net/gsr/1.0/polyline.json.

geometry/validPolyline |

**Example:**

{

"paths" : [

 [ [-97.06138,32.837], [-97.06133,32.836], [-97.06124,32.834], [-97.06127,32.832] ],

 [ [-97.06326,32.759], [-97.06298,32.755] ]

],

"spatialReference" : {"wkid" : 4326}

}

## Polygon

A polygon contains an array of rings and a spatial reference. Each ring is represented as an array of points and each point in the ring is represented as a two-element array. The 0 index is the x-coordinate and the 1 index is the y-coordinate.

|  |
| --- |
| * + 1. The JSON representation of a polygon SHALL validate against the JSON Schema http://schemas.opengis.net/gsr/1.0/polygon.json.

geometry/ validPolygon |

|  |
| --- |
| * + 1. The first ring SHALL be the exterior boundary of the polygon and use clockwise orientation, all other rings SHALL be interior boundaries and use counterclockwise orientation. All rings SHALL be closed, i.e. the first point of each ring is always the same as the last point.

geometry/rings |

**Example:**

{

 "rings" : [

 [[-100,30],[-100,50],[-60,50],[-60,30],[-100,30]],

 [[-83,35],[-74,35],[-74,41],[-83,41],[-83,35]]

 ],

 "spatialReference" : {"wkid" : 4326}

}

## Envelope

An envelope contains the corner points of an extent and is represented by xmin, ymin, xmax, and ymax, along with a spatial reference.

|  |
| --- |
| * + 1. The JSON representation of an envelope SHALL validate against the JSON Schema http://schemas.opengis.net/gsr/1.0/envelope.json.

geometry/validEnvelope |

**Example:**

{ "xmin" : -109.55, "ymin" : 25.76, "xmax" : -86.39, "ymax" : 49.94, "spatialReference" : {"wkid" : 4326} }

|  |
| --- |
| * + 1. xmin SHALL be smaller than or equal to xmax, ymin SHALL be smaller than or equal to ymax.

geometry/order |

## Geometry Array

An array of geometries contains a homogenuous collection of geometries. The first property specifies the geometry type, the second the array of geometry objects of that type.

|  |
| --- |
| * + 1. The JSON representation of a geometry arraySHALL validate against the JSON Schema http://schemas.opengis.net/gsr/1.0/geometries.json.

geometry/validArray |

**Example:**

{

"geometryType":"GeometryPoint",

"geometries":[{"x":-104.53,"y":34.74},{"x":-63.53,"y":10.23}],

"spatialReference" : {"wkid" : 4326}

}

|  |
| --- |
| * + 1. Each item in the "geometries" array SHALL validate against the schema of the geometry object stated in the property "geometryType", if the property is provided.

geometry/validItem |

# Features

## Overview

JSON representations of feature, object set and domain objects as well as layers and tables are used in several parts of the GeoServices REST API.

## Feature

A feature is an abstraction of real world phenomena. It is described by properties. In the GeoServices REST API, features may have a geometry property and they may additional have non-spatial attributes.

Non-spatial attribute information is organised in tables. Tables are made up of records and fields. Each field can store a specific type of data, such as a number, date, or piece of text.

A feature type is a collection of features that share the same geometry type (such as point, polyline, or polygon) and the same attribute fields. Streets, well points, parcels, soil types, and census tracts are examples of feature types.

In the JSON representation, a feature consists of two properties, geometry and attributes, which are both optional. The feature geometry can be any of the geometry objects.

EXAMPLE 1 A feature geometry with a point object in JSON:

"geometry" : {"x" : -118.15, "y" : 33.80}

The feature attributes are in a JSON object that contains a dictionary of name-value pairs. The names are the feature field names. The values are the field values, and they can be any of the standard JSON types: string, number, and boolean. Date values are encoded as numbers representing the milliseconds since epoch (January 1, 1970) in UTC.

EXAMPLE 2 The attributes of a feature in JSON:

"attributes" : {

 "OWNER" : "Joe Smith", //string

 "VALUE" : 94820.37, //number

 "APPROVED" : true, //boolean

 "LASTUPDATE" : 1227663551096 //date

}

**JSON Example**

{

 "geometry" : {"x" : -118.15, "y" : 33.80},

 "attributes" : {

 "OWNER" : "Joe Smith",

 "VALUE" : 94820.37,

 "APPROVED" : true,

 "LASTUPDATE" : 1227663551096

 }

}

|  |
| --- |
| * + 1. The JSON representation of a feature SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/feature.json**.

feature/valid |

|  |
| --- |
| * + 1. Each property in the "attributes" object SHALL have a unique name.

feature/uniqueAttributes |

In addition to feature geometry and atributes, a feature may also be related to other features. These relationships can be analysed in the Map Service and the Feature Service.

## Fields

Each field is characterised by several properties:

* name: The name of the field.
* alias: An optional alternate name for the field. Unlike the field's true name, an alias does not have to adhere to the limitations of the underlying datastore and may therefore contain spaces and special characters and start with a number.
* type: The data type of the field. Domains (see 10.7, 10.8, and 10.9) may additionally be used to limit the valid values of a field.
* length: For text fields this determines the maximum number of characters that may be input.

**JSON Example**

{

 "name" : "magnitude",

 "type" : "FieldTypeDouble",

 "alias" : "Magnitude"

}

Well-known values of the type property and their mapping to JSON Schema are:

|  |  |  |
| --- | --- | --- |
| **FieldTypeSmallInteger** | Short Integer. | JSON Schema: integer |
| **FieldTypeInteger** | Long Integer. | JSON Schema: integer |
| **FieldTypeSingle** | Single-precision floating-point number. | JSON Schema: number |
| **FieldTypeDouble** | Double-precision floating-point number. | JSON Schema: number |
| **FieldTypeString** | Character string. | JSON Schema: string |
| **FieldTypeDate** | Date. | JSON Schema: number |
| **FieldTypeOID** | Long Integer representing an object identifier. | JSON Schema: integer |
| **FieldTypeGeometry** | Geometry. | JSON Schema: object |
| **FieldTypeGUID, FieldTypeGlobalID** | Globally Unique Identifier. | JSON Schema: string |

## Feature Types, Layers and Tables

In the GeoServices REST API, feature types are organised in layers (for a feature type with a geometry field) and tables (for those without a geometry field). Layer/Table resources are used in the Map Service and the Feature Service.

A Layer/Table provides basic information about the layer or table such as its id, name, type (layer or type), and fields. It also provides information regarding the relationship of this feature type with other feature types. A feature type may be subdivided into subtypes, too.

For layers, a Layer/Table resource provides additional information related to the portrayal of the features in a map.

|  |
| --- |
| * + 1. The JSON representation of a Layer/Table resource SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/layerOrTable.json**.

feature/layerOrTableValid |

Examples of layers and tables are included in the Map Service and Feature Service standards.

## Feature Set

A feature set is a collection of features. All features are from the same feature type.

In addition to the collection of features in the "features" property, the fields of the feature type, its spatial reference and geometry type (for feature types with a geometry), and the field name of the object identifiers is provided.

**JSON Example**

{

 "objectIdFieldName" : "objectid",

 "globalIdFieldName" : "",

 "geometryType" : "GeometryPoint",

 "spatialReference" : {

 "wkid" : 4326

 },

 "fields" : [

 {

 "name" : "objectid",

 "type" : "FieldTypeOID",

 "alias" : "Object ID"

 },

 {

 "name" : "datetime",

 "type" : "FieldTypeDate",

 "alias" : "Earthquake Date",

 "length" : 36

 },

 {

 "name" : "depth",

 "type" : "FieldTypeDouble",

 "alias" : "Depth"

 },

 {

 "name" : "eqid",

 "type" : "FieldTypeString",

 "alias" : "Earthquake ID",

 "length" : 50

 },

 {

 "name" : "latitude",

 "type" : "FieldTypeDouble",

 "alias" : "Latitude"

 },

 {

 "name" : "longitude",

 "type" : "FieldTypeDouble",

 "alias" : "Longitude"

 },

 {

 "name" : "magnitude",

 "type" : "FieldTypeDouble",

 "alias" : "Magnitude"

 },

 {

 "name" : "numstations",

 "type" : "FieldTypeInteger",

 "alias" : "Number of Stations"

 },

 {

 "name" : "region",

 "type" : "FieldTypeString",

 "alias" : "Region",

 "length" : 200

 },

 {

 "name" : "source",

 "type" : "FieldTypeString",

 "alias" : "Source",

 "length" : 50

 },

 {

 "name" : "version",

 "type" : "FieldTypeString",

 "alias" : "Version",

 "length" : 50

 }

 ],

 "features" : [

 {

 "geometry" : {

 "x" : -178.24479999999991,

 "y" : 50.012500000000045

 },

 "attributes" : {

 "objectid" : 3745682,

 "datetime" : 1272210710000,

 "depth" : 31.100000000000001,

 "eqid" : "2010vma5",

 "latitude" : 50.012500000000003,

 "longitude" : -178.2448,

 "magnitude" : 4.7999999999999998,

 "numstations" : 112,

 "region" : "Andreanof Islands, Aleutian Islands, Alaska",

 "source" : "us",

 "version" : "Q"

 }

 },

 {

 "geometry" : {

 "x" : -72.865099999999927,

 "y" : -37.486599999999953

 },

 "attributes" : {

 "objectid" : 3745685,

 "datetime" : 1272210142999,

 "depth" : 40.600000000000001,

 "eqid" : "2010vma4",

 "latitude" : -37.486600000000003,

 "longitude" : -72.865099999999998,

 "magnitude" : 4.9000000000000004,

 "numstations" : 58,

 "region" : "Bio-Bio, Chile",

 "source" : "us",

 "version" : "7"

 }

 }

 ]

}

|  |
| --- |
| * + 1. The JSON representation of an Object Set SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/featureSet.json**.

feature/validFeatureSet |

## Feature ID Set

A feature identifier set is a collection of feature identifiers. All features are from the same feature type.

In addition, the field name of the identifier attribute in the features is identified.

**JSON Example**

{

"objectIdFieldName":"objectid",

"objectIds":[1,2,3,4,5,7]

}

|  |
| --- |
| * + 1. The JSON representation of an Object ID Set SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/featureIdSet.json**.

feature/validFeatureIdSet |

## Range Domain

Range domain specifies a range of valid numeric values for a field. The type property for range domains is range.

**JSON Example**

{

 "type": "range",

 "name": "Measured Length",

 "range": [1,10000]

}

|  |
| --- |
| * + 1. The JSON representation of a range domain SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/rangeDomain.json**.

feature/validRangeDomain |

## Coded Value Domain

Coded value domain specifies an explicit set of valid values for a field. Each valid value is assigned a unique name.

**JSON Example**

{

 "type": "codedValue",

 "name": "Material",

 "codedValues":

 [

 {

 "name": "Aluminum",

 "code": "AL"

 },

 {

 "name": "Copper",

 "code": "CU"

 },

 {

 "name": "Steel",

 "code": "STEL"

 },

 {

 "name": "Not Applicable",

 "code": "NA"

 }

 ]

}

The type property is fixed to "codedValue".

|  |
| --- |
| * + 1. The JSON representation of a coded value domain SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/codedValueDomain.json**.

feature/validCodedValueDomain |

|  |
| --- |
| * + 1. Each object in the "codedValues" object SHALL have a unique code and name.

feature/uniqueCode |

## Inherited Domain

Inherited domains apply to domains on subtypes of a feature type. An inherited domain implies that the domain for a field at the subtype level is the same as the domain for the field at the feature type (layer/table) level.

**JSON Example**

{

"type" : "inherited"

}

|  |
| --- |
| * + 1. The JSON representation of an inherited domain SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/inheritedDomain.json**.

feature/validInheritedDomain |

# Symbols, Labels, Renderers

## Overview

JSON representations of color, symbol, label and renderer objects are used in various parts of the GeoServices REST API.

## Color

Color is represented as a four-element array. The four elements represent values for red, green, blue, and alpha, in that order. Values range from 0 through 255.

**JSON Example**

[ 67, 0, 255, 40 ]

|  |
| --- |
| * + 1. The JSON representation of a color SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/color.json**.

symbol/validColor |

## Symbol Objects

### Overview

Symbols are graphics used to represent a feature in a map. Symbols can look like what they represent (trees, railroads, houses), or they can be abstract shapes (points, lines, polygons) or characters. Symbols are usually explained in a map legend.

The GeoServices REST API specifies several symbol objects and their JSON representation:

* Simple Marker Symbol: A simple symbol used to represent a point location on a map.
* Simple Line Symbol: A color or pattern used to represent polylines or the boundaries of polygons on a map.
* Simple Fill Symbol: A color or pattern used to fill polygons on a map.
* Picture Marker Symbol: Similar to Simple Marker Symbol, but supports the use of a graphic image.
* Picture Fill Symbol: Similar to Simple Fill Symbol, but supports the use of a graphic image to fill polygons.
* Text Symbol: A text style defined by font, size, character spacing, color, and so on, used to label maps and features.

### Simple Marker Symbol

Simple marker symbols can be used to symbolize point geometries.

**JSON Example**

{

 "type": "SMS",

 "style": "SMSSquare",

 "color": [76,115,0,255],

 "size": 8,

 "angle": 0,

 "xoffset": 0,

 "yoffset": 0,

 "outline":

 {

 "color": [152,230,0,255],

 "width": 1

 }

}

The type property is fixed to "SMS".

|  |
| --- |
| * + 1. The JSON representation of a Simple Marker Symbol SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/sms.json**.

symbol/validSMS |

The style property SHOULD be one of the following values:

|  |  |
| --- | --- |
| SMSCircle | The marker is a circle. |
| SMSSquare | The marker is a square. |
| SMSCross | The marker is a cross. |
| SMSX | The marker is an X. |
| SMSDiamond | The marker is a diamond. |

### Simple Line Symbol

Simple line symbols can be used to symbolize polyline geometries or outlines for polygon fills.

**JSON Example**

{

"type": "SLS",

"style": "SLSDot",

"color": [115,76,0,255],

"width": 1

}

The type property is fixed to "SLS".

|  |
| --- |
| * + 1. The JSON representation of a Simple Line Symbol SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/sls.json**.

symbol/validSLS |

The style property SHOULD be one of the following values:

|  |  |
| --- | --- |
| SLSSolid | The line is solid. |
| SLSDash | The line is dashed -------. |
| SLSDot | The line is dotted ....... |
| SLSDashDot | The line has alternating dashes and dots \_.\_.\_.\_. |
| SLSDashDotDot | The line has alternating dashes and double dots \_..\_..\_. |
| SLSNull | The line is invisible. |
| SLSInsideFrame | The line will fit into it's bounding rectangle, if any. |

### Simple Fill Symbol

Simple fill symbols can be used to symbolize polygon geometries.

**JSON Example**

{

 "type": "SFS",

 "style": "SFSSolid",

 "color": [115,76,0,255],

 "outline": {

 "type": "SLS",

 "style": "SLSSolid",

 "color": [110,110,110,255],

 "width": 1

 }

}

The type property is fixed to "SFS".

|  |
| --- |
| * + 1. The JSON representation of a Simple Fill Symbol SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/sfs.json**.

symbol/validSFS |

The style property SHOULD be one of the following values:

|  |  |
| --- | --- |
| SFSSolid | Solid fill. |
| SFSNull | Empty fill. |
| SFSHollow | Hollow fill (same as SFSNull). |
| SFSHorizontal | Horizontal hatch fill ------. |
| SFSVertical | Vertical hatch fill ||||||. |
| SFSForwardDiagonal | 45-degree downward, left-to-right hatch fill \\\. |
| SFSBackwardDiagonal | 45-degree upward, left-to-right hatch fill //////. |
| SFSCross | Horizontal and vertical crosshatch ++++++. |
| SFSDiagonalCross | 45-degree crosshatch xxxxxx. |

### Picture Marker Symbol

Picture marker symbols can be used to symbolize point geometries.

These symbols include the base64-encoded image data, as well as a URL that could be used to retrieve the image from the server.

**JSON Example**

{

 "type" : "PMS",

 "url" : "471E7E31",

 "imageData" : "iVBORw0KGgoAAAANSUhEUgAAABoAAAAaCAYAAACpSkzOAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADsQBlSsOGwAAAMNJREFUSIntlcENwyAMRZ+lSMyQFcI8rJA50jWyQuahKzCDT+6h0EuL1BA1iip8Qg/Ex99fYuCkGv5bKK0EcB40YgSE7bnTxsa58LeOnMd0QhwGXkxB3L0w0IDxPaMqpBFxjLMuaSVmRjurWIcRDHxaiWZuEbRcEhpZpSNhE9O81GiMN5E0ZRt2M0iVjshek8UkTQfZy8JqGHYP/rJhODD4T6wehtbB9zD0MPQwlOphaAxD/uPLK7Z8MB5gFet+WKcJPQDx29XkRhqr/AAAAABJRU5ErkJggg==",

 "contentType" : "image/png",

 "color" : null,

 "width" : 19.5,

 "height" : 19.5,

 "angle" : 0,

 "xoffset" : 0,

 "yoffset" : 0

}

The type property is fixed to "PMS".

|  |
| --- |
| * + 1. The JSON representation of a Picture Marker Symbol SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/pms.json**.

symbol/validPMS |

|  |
| --- |
| * + 1. The value of the "imageData" property SHALL be a base64-encoded content of a file where the file SHALL be of the content type specified in the "contentType" property.

symbol/base64PMS |

|  |
| --- |
| * + 1. The "contentType" property SHALL be a valid MIME type. At least "image/png" SHALL be supported.

symbol/contentTypePMS |

|  |
| --- |
| * + 1. The file referenced from the URL in the property "url" SHALL exits and be the same as the file provided in the "imageData" property. If the URL is relative, it is realtive to the URL of the JSON document containing the symbol.

symbol/urlPMS |

### Picture Fill Symbol

Picture fill symbols can be used to symbolize polygon geometries.

These symbols include the base64-encoded image data, as well as a URL that could be used to retrieve the image from the server.

**JSON Example**

{

 "type" : "PFS",

 "url" : "866880A0",

 "imageData" : "iVBORw0KGgoAAAANSUhEUgAAAFQAAABUCAYAAAAcaxDBAAAAAXNSR0IB2cksfwAAAAlwSFlzAAAOxAAADsQBlSsOGwAAAM9JREFUeJzt0EEJADAMwMA96l/zTBwUSk5ByLxQsx1wTUOxhmINxRqKNRRrKNZQrKFYQ7GGYg3FGoo1FGso1lCsoVhDsYZiDcUaijUUayjWUKyhWEOxhmINxRqKNRRrKNZQrKFYQ7GGYg3FGoo1FGso1lCsoVhDsYZiDcUaijUUayjWUKyhWEOxhmINxRqKNRRrKNZQrKFYQ7GGYg3FGoo1FGso1lCsoVhDsYZiDcUaijUUayjWUKyhWEOxhmINxRqKNRRrKNZQrKFYQ7GGYh/hIwFRFpnZNAAAAABJRU5ErkJggg==",

 "contentType" : "image/png",

 "color" : null,

 "outline" :

 {

 "type" : "SLS",

 "style" : "SLSSolid",

 "color" : [110,110,110,255],

 "width" : 1

 },

 "width" : 63,

 "height" : 63,

 "angle" : 0,

 "xoffset" : 0,

 "yoffset" : 0,

 "xscale" : 1,

 "yscale" : 1

}

The type property is fixed to "PFS".

|  |
| --- |
| * + 1. The JSON representation of a Picture Fill Symbol SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/pfs.json**.

symbol/validPFS |

|  |
| --- |
| * + 1. The value of the "imageData" property SHALL be a base64-encoded content of a file where the file SHALL be of the content type specified in the "contentType" property.

symbol/base64PFS |

|  |
| --- |
| * + 1. The "contentType" property SHALL be a valid MIME type. At least "image/png" SHALL be supported.

symbol/contentTypePFS |

|  |
| --- |
| * + 1. The file referenced from the URL in the property "url" SHALL exits and be the same as the file provided in the "imageData" property. If the URL is relative, it is realtive to the URL of the JSON document containing the symbol.

symbol/urlPFS |

### Text Symbol

Text symbols are used to add text to a feature (labeling).

**JSON Example**

{

 "type": "TS",

 "color": [78,78,78,255],

 "backgroundColor": null,

 "borderLineColor": null,

 "verticalAlignment": "bottom",

 "horizontalAlignment": "left",

 "rightToLeft": false,

 "angle": 0,

 "xoffset": 0,

 "yoffset": 0,

 "font": {

 "family": "Arial",

 "size": 12,

 "style": "normal",

 "weight": "bold",

 "decoration": "none"

 }

}

The type property is fixed to "TS".

|  |
| --- |
| * + 1. The JSON representation of a Text Symbol SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/ts.json**.

symbol/validTS |

The verticalAlignment property SHOULD be one of the following values: "baseline", "top", "middle", or "bottom".

The horizontalAlignment property SHOULD be one of the following values: "left", "right", "center", or "justify".

The style property in the "font" object SHOULD be one of the following values: "italic", "normal", or "oblique".

The weight property in the "font" object SHOULD be one of the following values: "bold", "bolder", "lighter", or "normal".

The decoration property in the "font" object SHOULD be one of the following values: "line-through", "underline", or "none".

## Label Objects

### Overview

Labels are descriptive text, usually based on one or more feature attributes. In maps, labels are placed dynamically on or near features based on user-defined rules and in response to changes in the map display. Label placement rules and display properties are defined for an entire layer.

The GeoServices REST API specifies two objects related to labels and their JSON representation:

* Label Class: a category of labels that represents features with the same labeling properties. For example, in a roads layer, label classes could be created to define information and style for each type of road: interstate, state highway, county road, and so on.
* Labeling Info: specifies the label definition for a layer, an array of label classes.

### Label Class

A label class specifies the label definition for a given scale range.

**JSON Example**

{

 "labelPlacement": "ServerPointLabelPlacementAboveRight",

 "labelExpression": "[NAME]",

 "useCodedValues": false,

 "symbol": {

 "type": "TS",

 "color": [38,115,0,255],

 "backgroundColor": null,

 "borderLineColor": null,

 "verticalAlignment": "bottom",

 "horizontalAlignment": "left",

 "rightToLeft": false,

 "angle": 0,

 "xoffset": 0,

 "yoffset": 0,

 "font": {

 "family": "Arial",

 "size": 11,

 "style": "normal",

 "weight": "bold",

 "decoration": "none"

 }

 },

 "minScale": 0,

 "maxScale": 0

}

|  |
| --- |
| * + 1. The JSON representation of a Label Class SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/label.json**.

symbol/validLabel |

Label placement is represented as a literal string. It specifies the placement of the label with respect to that of its feature. Below is a list of label placement values categorized by the geometry type of the feature.

The labelPlacement property for a point feature SHOULD be one of the following values:

* ServerPointLabelPlacementAboveCenter
* ServerPointLabelPlacementAboveLeft
* ServerPointLabelPlacementAboveRight
* ServerPointLabelPlacementBelowCenter
* ServerPointLabelPlacementBelowLeft
* ServerPointLabelPlacementBelowRight
* ServerPointLabelPlacementCenterCenter
* ServerPointLabelPlacementCenterLeft
* ServerPointLabelPlacementCenterRight

The labelPlacement property for a line feature SHOULD be one of the following values:

* ServerLinePlacementAboveAfter
* ServerLinePlacementAboveAlong
* ServerLinePlacementAboveBefore
* ServerLinePlacementAboveEnd
* ServerLinePlacementAboveStart
* ServerLinePlacementBelowAfter
* ServerLinePlacementBelowAlong
* ServerLinePlacementBelowBefore
* ServerLinePlacementBelowEnd
* ServerLinePlacementBelowStart
* ServerLinePlacementCenterAfter
* ServerLinePlacementCenterAlong
* ServerLinePlacementCenterBefore
* ServerLinePlacementCenterEnd
* ServerLinePlacementCenterStart

The labelPlacement property for a polygon feature SHOULD be one of the following values:

* ServerPolygonPlacementAlwaysHorizontal

### Labeling Info

The labeling info object specifies the label definition for a layer. It is expressed as an array of label classes.

**JSON Example**

[

 {

 "labelPlacement": "ServerPolygonPlacementAlwaysHorizontal",

 "labelExpression": "[TAG]",

 "useCodedValues": false,

 "symbol": {

 "type": "TS",

 "color": [78,78,78,255],

 "backgroundColor": null,

 "borderLineColor": null,

 "verticalAlignment": "bottom",

 "horizontalAlignment": "left",

 "rightToLeft": false,

 "angle": 0,

 "xoffset": 0,

 "yoffset": 0,

 "font": {

 "family": "Arial",

 "size": 12,

 "style": "normal",

 "weight": "bold",

 "decoration": "none"

 }

 },

 "minScale": 1999,

 "maxScale": 0

 },

 {

 "labelPlacement": "ServerPolygonPlacementAlwaysHorizontal",

 "labelExpression": "[TAG]",

 "useCodedValues": true,

 "symbol": {

 "type": "TS",

 "color": [78,78,78,255],

 "backgroundColor": null,

 "borderLineColor": null,

 "verticalAlignment": "bottom",

 "horizontalAlignment": "left",

 "rightToLeft": false,

 "angle": 0,

 "xoffset": 0,

 "yoffset": 0,

 "font": {

 "family": "Arial",

 "size": 12,

 "style": "normal",

 "weight": "bold",

 "decoration": "none"

 }

 },

 "minScale": 0,

 "maxScale": 7100

 }

]

|  |
| --- |
| * + 1. The JSON representation of a Labeling Info SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/labelInfo.json**.

symbol/validLabelInfo |

## Renderer Objects

### Overview

Renderer objects describe how data appears when displayed.

The GeoServices REST API specifies three relatively simple, but commonly used renderers and their JSON representation:

* Simple Renderer: A fixed symbol.
* Unique Value Renderer: Symbols are selected based on matching field values, e.g. coded values.
* Class Breaks Renderer: Symbols are selected based on ranges of some numeric field.

### Simple Renderer

A simple renderer uses one symbol only.

**JSON Example**

{

 "type": "simple",

 "symbol":

 {

 "type": "SMS",

 "style": "SMSCircle",

 "color": [255,0,0,255],

 "size": 5,

 "angle": 0,

 "xoffset": 0,

 "yoffset": 0,

 "outline":

 {

 "color": [0,0,0,255],

 "width": 1

 }

 },

 "label": "",

 "description": ""

}

The type property is fixed to "simple".

|  |
| --- |
| * + 1. The JSON representation of a Simple Renderer SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/simpleRenderer.json**.

symbol/validSR |

### Unique Value Renderer

A unique value renderer symbolizes groups of features that have matching field values.

**JSON Example**

{

 "type" : "uniqueValue",

 "field1" : "SubtypeCD",

 "field2" : null,

 "field3" : null,

 "fieldDelimiter" : ", ",

 "defaultSymbol" :

 {

 "type" : "SLS",

 "style" : "SLSSolid",

 "color" : [130,130,130,255],

 "width" : 1

 },

 "defaultLabel" : "\u003Other values\u003e",

 "uniqueValueInfos" : [

 {

 "value" : "1",

 "label" : "Duct Bank",

 "description" : "Duct Bank description",

 "symbol" :

 {

 "type" : "SLS",

 "style" : "SLSDash",

 "color" : [76,0,163,255],

 "width" : 1

 }

 },

 {

 "value" : "2",

 "label" : "Trench",

 "description" : "Trench description",

 "symbol" :

 {

 "type" : "SLS",

 "style" : "SLSDot",

 "color" : [115,76,0,255],

 "width" : 1

 }

 }

 ]

}

The type property is fixed to "uniqueValue".

|  |
| --- |
| * + 1. The JSON representation of a Unique Value Renderer SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/uniqueValueRenderer.json**.

symbol/validUVR |

### Class Breaks Renderer

A class breaks renderer symbolizes each feature based on the value of some numeric field.

**JSON Example**

{

 "type" : "classBreaks",

 "field" : "Shape.area",

 "minValue" : 10.3906320193541,

 "classBreakInfos" : [

 {

 "classMaxValue" : 1000,

 "label" : "10.0 - 1000.000000",

 "description" : "10 to 1000",

 "symbol" :

 {

 "type" : "SFS",

 "style" : "SFSSolid",

 "color" : [236,252,204,255],

 "outline" :

 {

 "type" : "SLS",

 "style" : "SLSSolid",

 "color" : [110,110,110,255],

 "width" : 0.4

 }

 }

 },

 {

 "classMaxValue" : 5000,

 "label" : "1000.000001 - 5000.000000",

 "description" : "1000 to 5000",

 "symbol" :

 {

 "type" : "SFS",

 "style" : "SFSSolid",

 "color" : [218,240,158,255],

 "outline" :

 {

 "type" : "SLS",

 "style" : "SLSSolid",

 "color" : [110,110,110,255],

 "width" : 0.4

 }

 }

 }

 ]

}

The type property is fixed to "classBreaks".

|  |
| --- |
| * + 1. The JSON representation of a Class Breaks Renderer SHALL validate against the JSON Schema **http://schemas.opengis.net/gsr/1.0/classBreaksRenderer.json**.

symbol/validCBR |

Annex A
(normative)

Abstract Test Suite

Conformance class: core

* 1. Test: core/get

|  |  |
| --- | --- |
| Requirements | **core/get** |
| Test purpose | Verify that GET requests are safe and idempotent. |
| Test method | Inspect the documentation to identify, if requests specified in the GeoServices REST API standard that support the HTTP GET method, are all safe and idempotent as specified in HTTP (RFC 2616, section 9.1). |
| Test type | Capability |

* 1. Test: core/extensibility

|  |  |
| --- | --- |
| Requirements | **core/unknownParameter** |
| Test purpose | Verify that the service supports the extensibility requirements. |
| Test method | For each service supported by a product, construct various requests with additional parameters not specified in the current standards and send these to the server. Inspect the responses and confirm that the implementation has ignored the additional parameters. |
| Test type | Capability |

|  |  |
| --- | --- |
| Requirements | **core/unknownParameterValue** |
| Test purpose | Verify that the service detects unsupported, but requested extensions |
| Test method | For each service supported by a product, construct various requests with parameter values not specified in the current standards and send these to the server. Inspect the responses and confirm that the implementation has raised exceptions. |
| Test type | Capability |

* 1. Test: core/exceptions

|  |  |
| --- | --- |
| Requirements | **core/exception, core/exceptionCode, core/exceptionValid** |
| Test purpose | Verify that the service supports the exception requirements. |
| Test method | For each service that the product supports, construct valid and invalid requests.Inspect the responses and verify that all valid requests return no exception or an exception with a 5xx code.Verify that all invalid requests return an exception and that the error code is the appropriate HTTP status code. Verify that the response validates against http://schemas.opengis.net/gsr/1.0/exception.json.  |
| Test type | Capability |

Conformance class: jsonp

* 1. Test: core/exceptions

|  |  |
| --- | --- |
| Requirements | **jsonp/callback** |
| Test purpose | Verify that the service supports the JSONP requirements. |
| Test method | For each service that the product supports, construct valid requests and execute it twice, once with a parameter callback with a value of "test" and once without.Inspect the responses and verify that response of the request with the callback parameter are "test(<ResponseWithoutCallbackParameter>);". |
| Test type | Capability |

Conformance class: geometry

* 1. Test: geometry/sr

|  |  |
| --- | --- |
| Requirements | **geometry/validSR, geometry/wkidXorWkt, geometry/wkid, geometry/wkt** |
| Test purpose | Verify that every spatial reference object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a spatial reference object verify that the object* validates against http://schemas.opengis.net/gsr/1.0/spatialreference.json
* contains either a "wkid" or a "wkt" property

If it contains a "wkid" property, verify that the list of coordinate reference systems in http://schemas.opengis.net/gsr/1.0/sr.json contains an object with the same "wkid" property.If it contains a "wkt" property, verify the content against the requirements in OGC Simple Feature Access – Part 1: Common Architecture, section 6.4. |
| Test type | Capability |

* 1. Test: geometry/point

|  |  |
| --- | --- |
| Requirements | **geometry/validPoint** |
| Test purpose | Verify that every point object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a point object verify that the object validates against http://schemas.opengis.net/gsr/1.0/point.json. |
| Test type | Capability |

* 1. Test: geometry/multipoint

|  |  |
| --- | --- |
| Requirements | **geometry/validMultipoint** |
| Test purpose | Verify that every multi point object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a multi point object verify that the object validates against http://schemas.opengis.net/gsr/1.0/multipoint.json. |
| Test type | Capability |

* 1. Test: geometry/polyline

|  |  |
| --- | --- |
| Requirements | **geometry/polyline** |
| Test purpose | Verify that every polyline object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a polyline object verify that the object validates against http://schemas.opengis.net/gsr/1.0/polyline.json. |
| Test type | Capability |

* 1. Test: geometry/polygon

|  |  |
| --- | --- |
| Requirements | **geometry/validPolygon, geometry/rings** |
| Test purpose | Verify that every polygon object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a polygon object verify that the object validates against http://schemas.opengis.net/gsr/1.0/polygon.json.Verify that all rings have identical start and end points.Verify that the orientation of the first ring is clockwise and that the orientation of all other rings is counterclockwise. |
| Test type | Capability |

* 1. Test: geometry/envelope

|  |  |
| --- | --- |
| Requirements | **geometry/validEnvelope, geometry/order** |
| Test purpose | Verify that every envelope object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of an envelope object verify that the object validates against http://schemas.opengis.net/gsr/1.0/envelope.json and that xmin is smaller than or equal to xmax and that ymin is smaller than or equal to ymax. |
| Test type | Capability |

* 1. Test: geometry/array

|  |  |
| --- | --- |
| Requirements | **geometry/validArray, geometry/validItem** |
| Test purpose | Verify that every array object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a geometry array object verify that the object validates against http://schemas.opengis.net/gsr/1.0/geometries.json. |
| Test type | Capability |

Conformance class: feature

* 1. Test: feature/feature

|  |  |
| --- | --- |
| Requirements | **feature/valid, feature/uniqueAttributes** |
| Test purpose | Verify that every feature object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a feature object verify that * the object validates against http://schemas.opengis.net/gsr/1.0/feature.json
* each property in the "attributes" object has a unique name.
 |
| Test type | Capability |

* 1. Test: feature/layerOrTable

|  |  |
| --- | --- |
| Requirements | **feature/layerOrTableValid** |
| Test purpose | Verify that every feature type description meets the requirements. |
| Test method | Inspect the JSON representation and for each layer/table object verify that the object validates against http://schemas.opengis.net/gsr/1.0/layerOrTable.json. |
| Test type | Capability |

* 1. Test: feature/featureSet

|  |  |
| --- | --- |
| Requirements | **feature/validFeatureSet** |
| Test purpose | Verify that every feature collection meets the requirements. |
| Test method | Inspect the JSON representation and for each feature collection object verify that the object validates against http://schemas.opengis.net/gsr/1.0/featureSet.json. |
| Test type | Capability |

* 1. Test: feature/featureIdSet

|  |  |
| --- | --- |
| Requirements | **feature/validFeatureIdSet** |
| Test purpose | Verify that every feature identifier collection meets the requirements. |
| Test method | Inspect the JSON representation and for each feature identifier collection object verify that the object validates against http://schemas.opengis.net/gsr/1.0/featureIdSet.json. |
| Test type | Capability |

* 1. Test: feature/rangeDomain

|  |  |
| --- | --- |
| Requirements | **feature/validRangeDomain** |
| Test purpose | Verify that every range domain meets the requirements. |
| Test method | Inspect the JSON representation and for each range domain object verify that the object validates against http://schemas.opengis.net/gsr/1.0/rangeDomain.json. |
| Test type | Capability |

* 1. Test: feature/codedValueDomain

|  |  |
| --- | --- |
| Requirements | **feature/validCodedValueDomain, feature/uniqueCode** |
| Test purpose | Verify that every coded value domain meets the requirements. |
| Test method | Inspect the JSON representation and for each coded value domain object verify that * the object validates against http://schemas.opengis.net/gsr/1.0/codedValueDomain.json
* each object in the "codedValues" object has a unique code and name.
 |
| Test type | Capability |

* 1. Test: feature/inheritedDomain

|  |  |
| --- | --- |
| Requirements | **feature/validInheritedDomain** |
| Test purpose | Verify that every inherited domain meets the requirements. |
| Test method | Inspect the JSON representation and for each inherited domain object verify that the object validates against http://schemas.opengis.net/gsr/1.0/inheritedDomain.json. |
| Test type | Capability |

Conformance class: symbol

* 1. Test: symbol/color

|  |  |
| --- | --- |
| Requirements | **symbol/validColor** |
| Test purpose | Verify that every color array meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a color array verify that the array validates against http://schemas.opengis.net/gsr/1.0/color.json. |
| Test type | Capability |

* 1. Test: symbol/sms

|  |  |
| --- | --- |
| Requirements | **symbol/validSMS** |
| Test purpose | Verify that every simple marker symbol object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a symbol verify that the object validates against http://schemas.opengis.net/gsr/1.0/sms.json. |
| Test type | Capability |

* 1. Test: symbol/sls

|  |  |
| --- | --- |
| Requirements | **symbol/validSLS** |
| Test purpose | Verify that every simple line symbol object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a symbol verify that the object validates against http://schemas.opengis.net/gsr/1.0/sls.json. |
| Test type | Capability |

* 1. Test: symbol/sfs

|  |  |
| --- | --- |
| Requirements | **symbol/validSFS** |
| Test purpose | Verify that every simple fill symbol object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a symbol verify that the object validates against http://schemas.opengis.net/gsr/1.0/sfs.json. |
| Test type | Capability |

* 1. Test: symbol/pms

|  |  |
| --- | --- |
| Requirements | **symbol/validPMS, symbol/base64PMS, symbol/contentTypePMS, symbol/urlPMS** |
| Test purpose | Verify that every picture marker symbol object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a symbol verify that * the object validates against http://schemas.opengis.net/gsr/1.0/pms.json
* the "contentType" property is a valid MIME type
* the value of the "imageData" property is a base64-encoded content of a file
* that file is of the content type specified in the "contentType" property
* the file referenced from the URL in the property "url" exits and is identical to the file after decoding the "imageData" property.

Inspect the documentation to verify that the content type "image/png" is supported. |
| Test type | Capability |

* 1. Test: symbol/pfs

|  |  |
| --- | --- |
| Requirements | **symbol/validPFS, symbol/base64PFS, symbol/contentTypePFS, symbol/urlPFS** |
| Test purpose | Verify that every picture fill symbol object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a symbol verify that * the object validates against http://schemas.opengis.net/gsr/1.0/pfs.json
* the "contentType" property is a valid MIME type
* the value of the "imageData" property is a base64-encoded content of a file
* that file is of the content type specified in the "contentType" property
* the file referenced from the URL in the property "url" exits and is identical to the file after decoding the "imageData" property.

Inspect the documentation to verify that the content type "image/png" is supported. |
| Test type | Capability |

* 1. Test: symbol/ts

|  |  |
| --- | --- |
| Requirements | **symbol/validTS** |
| Test purpose | Verify that every text symbol object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a symbol verify that the object validates against http://schemas.opengis.net/gsr/1.0/ts.json. |
| Test type | Capability |

* 1. Test: symbol/label

|  |  |
| --- | --- |
| Requirements | **symbol/validLabel** |
| Test purpose | Verify that every label class object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a label class object verify that the object validates against http://schemas.opengis.net/gsr/1.0/label.json. |
| Test type | Capability |

* 1. Test: symbol/labelInfo

|  |  |
| --- | --- |
| Requirements | **symbol/validLabelInfo** |
| Test purpose | Verify that every labeling info object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of a labeling info object verify that the object validates against http://schemas.opengis.net/gsr/1.0/labelInfo.json. |
| Test type | Capability |

* 1. Test: symbol/simpleRenderer

|  |  |
| --- | --- |
| Requirements | **symbol/validSR** |
| Test purpose | Verify that every simple renderer object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a renderer object verify that the object validates against http://schemas.opengis.net/gsr/1.0/simpleRenderer.json. |
| Test type | Capability |

* 1. Test: symbol/uniqueValueRenderer

|  |  |
| --- | --- |
| Requirements | **symbol/validUVR** |
| Test purpose | Verify that every unique value renderer object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a renderer object verify that the object validates against http://schemas.opengis.net/gsr/1.0/uniqueValueRenderer.json. |
| Test type | Capability |

* 1. Test: symbol/classBreaksRenderer

|  |  |
| --- | --- |
| Requirements | **symbol/validCBR** |
| Test purpose | Verify that every class breaks renderer object meets the requirements. |
| Test method | Inspect the JSON representation and for each occurance of such a renderer object verify that the object validates against http://schemas.opengis.net/gsr/1.0/classBreaksRenderer.json. |
| Test type | Capability |

Annex B
(normative)

Parameter syntax rules

The syntax of parameter values that have to be supported by compliant implementations is specified using the Augmented BNF for Syntax Specifications (ABNF, IETF RFC 5234).

The basic rules used in parameter tables in this standard and other standards of the GeoServices REST API are specified in this Annex.

The following core rules are from RFC 5234:

ALPHA = %x41-5A / %x61-7A ; A-Z / a-z

BIT = "0" / "1"

DIGIT = %x30-39 ; 0-9

VCHAR = %x21-7E ; visible (printing) characters

The following rules are specified by this standard:

NUMBER = ["-"] 1\*DIGIT ["." \*DIGIT]

DIGIT19 = ("1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9")

INTEGER = ["-"] 1\*DIGIT

POSINT = DIGIT19 \*DIGIT

XMIN = NUMBER

YMIN = NUMBER

XMAX = NUMBER

YMAX = NUMBER

WIDTH = POSINT

HEIGHT = POSINT

NAME = ALPHA \*(ALPHA / DIGIT / "-")

STRING = \*VCHAR

QUOTEDSTRING = "'" STRING "'"

STRINGARRAY = "[" QUOTEDSTRING \*("," QUOTEDSTRING) "]"

FORMATNAME = NAME

LAYERID = POSINT

BOOLEAN = "true" / "false"

NUMID = 1\*DIGIT

JSON = "{" STRING "}"

EXPR = OREXPR

OREXPR = ANDEXPR \*("OR" ANDEXPR)

ANDEXPR = EQEXPR \*("AND" EQEXPR)

EQEXPR = RELEXPR \*(("=" / "<>") RELEXPR)

RELEXPR = ADDEXPR [("<" / ">" / "<=" / ">=") ADDEXPR]

ADDEXPR = MULEXPR \*(("+" / "-" ) MULEXPR)

MULEXPR = PRIMEXPR \*(("\*" / "/" ) PRIMEXPR)

PRIMEXPR = NUMBER / "'" STRING "'" / BOOLEAN / "(" EXPR ")"

Bibliography

D. Crocker, P. Overell, "Augmented BNF for Syntax Specifications: ABNF", IETF RFC 5234, January 2008

Subbu Allamaraju, "RESTful Web Services Cookbook", O'Reilly Media / Yahoo Press, February 2010

1. [www.opengeospatial.org/cite](http://www.opengeospatial.org/cite) [↑](#footnote-ref-1)
2. See Clause 6 for a discussion about the use of HTTP methods by this standard. [↑](#footnote-ref-2)
3. see <http://en.wikipedia.org/wiki/JSONP> [↑](#footnote-ref-3)
4. see <http://www.w3.org/TR/html4/interact/forms.html#h-17.13> [↑](#footnote-ref-4)
5. see <https://www.w3.org/Bugs/Public/show_bug.cgi?id=10671> [↑](#footnote-ref-5)
6. see [http://www.silverlight.net/learn/data-networking/network-services-(soap,-rest-and-more)/rest-services-(silverlight-quickstart)](http://www.silverlight.net/learn/data-networking/network-services-%28soap%2C-rest-and-more%29/rest-services-%28silverlight-quickstart%29) [↑](#footnote-ref-6)
7. see <http://help.adobe.com/en_US/Flex/4.0/AccessingData/WSbde04e3d3e6474c46c45e7b4120d413dc14-8000.html#WSbde04e3d3e6474c46b5600a121e707ef97-8000> and <http://help.adobe.com/en_US/Flex/4.0/AccessingData/WSbde04e3d3e6474c4-668f02f4120d422cf08-7ffe.html#WSbde04e3d3e6474c4-668f02f4120d422cf08-7ff9> [↑](#footnote-ref-7)
8. See also the related discussion in the section „Future Work“ of this document. [↑](#footnote-ref-8)